EUROMUSE: A web-based system for the management of MUSEum objects and their interoperability with EUROpeana

by

Giannis Skevakis, Varvara Kalokyri

A thesis submitted to the

Department of Electronic & Computer Engineering

of the Technical University of Crete

in partial fulfillment of the

requirements for the Diploma of

Electronic Engineer and Computer Engineer

advisor: Prof. Stavros Christodoulakis

Chania, August 2011

Abstract

The continuing acceleration in the digitization of information is causing the traditional model of museums to expand to include high-resolution images of their collections for perusal, study, and exploration from any place with Internet connectivity. However, an impressive abundance of high quality digital content that is available in museums remains largely unexploited due to the lack of interconnection and interoperability among the management systems of museums, the lack of centralized access through a European point of reference like Europeana, as well as the inefficiency of current content organization and the metadata used. In this thesis we present EuroMuse: a web-based management system for museums, archives and digital collections, which facilitates the authoring and metadata enrichment of cultural heritage objects. It is a multilingual tool, that establishes interoperability between museums and Europeana, through its metadata-importing module, which enables the seamless transportation of legacy metadata into the system, supporting a rich metadata element set, which includes the Europeana Semantic Elements (ESE). Moreover, it supports the semantic linkage of the cultural heritage objects with well-established controlled vocabularies. It is developed with the Google Web Toolkit (GWT) framework, in the context of Natural Europe project and is actively being used by the cultural museum experts of six European Natural History Museums who have evaluated the system and have already described over 1000 fully described cultural heritage objects.

Acknowledgements

We would like to thank our supervisor, Prof. Stavros Christodoulakis, for his encouragement and his continuous guidance and support throughout our research. We would like also to thank him for the important experiences he offered us during our stay at the Laboratory of Distributed Multimedia Information Systems and Applications (MUSIC).

We would like to express our gratitude to the readers of this thesis Mr Antonios Deligiannakis and Mrs Aikaterini Mania for the time they devoted and their critical evaluation.

Our appreciation goes to Nektarios Gioldasis and Fotis Kazasis for their supervision and their valuable help regarding this thesis. We are also grateful to Konstantinos Makris for being always ready to offer his help whenever needed.

We would also thank the rest staff of the Laboratory for the pleasant environment they provided.

Giannis Skevakis, Varvara Kalokyri

Technical University of Crete

July 2011

Table Of Contents

List Of Tables

List Of Figures

Chapter 1

Introduction

The continuing acceleration in the digitization of information, combined with the increasing capacity of digital information storage, is causing the expansion of the traditional model of museums (i.e. as static “collections of collections” of three-dimensional specimens and artifacts) to include virtual exhibits and high-resolution images of their collections for perusal, study, and exploration from any place with Internet connectivity. However, an impressive abundance of high quality digital content that is available in museums remains largely unexploited due to a number of barriers such as: the lack of interconnection and interoperability between the management systems of museums, the lack of centralized access through a European point of reference like Europeana, as well as the inefficiency of current content organization and the metadata used.

In the wide public, Europeana [1] is primarily perceived as a portal exposing increasingly impressive amounts of cultural heritage from various sources to Europe's citizens. However, Europeana is not a Web Portal, but a services platform, providing an Application Program Interface (API) enabling cultural institutions and users to access/provide content to Europeana and build applications using Europeana functionalities for their own use.

The main purpose of Europeana is to enable people to explore the digital resources of Europe's museums, libraries, archives and audio-visual collections. It promotes discovery and networking opportunities in a multilingual space where users can engage, share in and be inspired by the rich diversity of Europe's cultural and scientific heritage.

Ideas and inspiration can be found within the more than 15 million items on Europeana. These objects include:

- Images - paintings, drawings, maps, photos and pictures of museum objects
- Texts - books, newspapers, letters, diaries and archival papers
- Sounds - music and spoken word from cylinders, tapes, discs and radio broadcasts
- Videos - films, newsreels and TV broadcasts

On a very abstract level, Europeana can be seen as a large collection of surrogate objects representing born digital or digitized cultural heritage objects which themselves remain outside the Europeana data space.

The system presented in this thesis is the first step towards allowing the connection of digital collections with Europeana. It is a web-based management system for museums, archives and digital collections, which facilitates the authoring and metadata enrichment of cultural heritage objects.

EuroMuse is a multilingual tool, which establishes the interoperability between museums and Europeana and the seamless ingestion of legacy metadata, through its metadata-importing module. It supports a rich metadata element set, which is a superset of the Europeana Semantic Elements (ESE) metadata format as well as a variety of the most popular media formats. The main features of EuroMuse include the publication of multimedia objects as well as the semantic linkage of the objects with well-established controlled vocabularies and the real-time collaboration among end-users with concurrency control mechanisms. The client side developed with Google Web Toolkit (GWT), the state of the art framework for writing complex RIA applications.

Finally, EuroMuse is developed in the context of the Natural Europe project and is actively being used by the cultural museum experts of six European Natural History Museums who have evaluated the system and have already described over 1000 fully described cultural heritage objects (CHOs) organized in CHO collections. Until the end of this project, 16.000 CHOs will be described through our application.

The six museums that are already using our application are:

- University of Crete - Natural History Museum of Crete – GREECE
- National Museum of Natural History – University of Lisbon – PORTUGAL
- Jura-Museum Eichstaett – GERMANY
- Arctic-Center – FINLAND
- The Estonian Museum of Natural History – ESTONIA
- Hungarian Natural History Museum – HUNGARY

Chapter 2

Background technologies

In this chapter we present the standards used in this thesis, as well as the technologies used for the implementation of our application. Firstly, in Section 2.1 we present the Europeana Semantic Elements, which is the main metadata vocabulary used by Europeana to describe museum objects. Section 2.2 describes AJAX, the leading technology in Web 2.0. Sections 2.3, 2.4, 2.5 and 2.6 talk about Java Servlet technology, XML,XML Schema and XML Beans respectively, which are the main technologies used for the development of EuroMuse. Section 2.7 describes Apache Lucene and Solr used by EuroMuse for indexing and fast retrieval of vocabulary terms, followed by RDF (Section 2.8) and SKOS (Section 2.9), the formats that the vocabularies are expressed. Lastly, in Section 2.10 we present Google Web Toolkit, which was used to implement the client side of the system.

2.1. The Europeana Semantic Elements (ESE)

Europeana's main goal is to provide integrated access to digital objects from the cultural heritage organizations of all the nations of the European Union and make them discoverable together in a common on-line environment. To achieve that, it needs to harvest and index the descriptive metadata associated with the digital objects. As there is no universal metadata standard applied across the participating domains, a set of metadata elements has been developed that will allow a common set of information to be supplied to support the functionality desired by the user and needed for the operation of the underlying system.

The Europeana Semantic Elements (ESE) [2] (latest version 3.4, March 2011) is an updated version of the metadata set that has been used from the start in the Europeana prototype in November 2008. This version has been updated to take account of the Data Quality Improvement Plan which makes more elements mandatory and changes the usage of some elements. It is an application profile based on Dublin Core [3], providing a generic set of terms that can be applied to heterogeneous materials thereby providing a baseline to allow contributors to take advantage of their existing rich descriptions.

To provide ESE-compliant metadata, it is necessary for contributors to map elements from their own metadata format to ESE. In addition to the mapping, it is necessary for a normalisation process to be carried out on some values to enable machine readability. To this end, an XML Schema [4] has also been produced (building on Dublin Core and Dublin Core Terms schemas) as a further tool to assist providers in ensuring compliance with ESE. The ESE specification defines an extensive list of elements. However, it classifies them according to their importance in achieving a common basis for answering “who, what, where and when” questions.

Below, are listed all the elements of the Europeana Semantics Elements (version 3.4.) with a brief description of each element.

Mandatory Elements	
dc:title	The title or name by which the digital object is known.
dc:description	A prose description of the digital object.
dc:language	This element is used to state the language of the digital object and is repeated if the object has more than one language.
europeana:dataProvider	The names of the organisations who supply the data to an aggregator to be unambiguously recorded.
europeana:isShownAt	This element contains a URL where the object is displayed within an information context or is accessed indirectly via another link.
europeana:isShownBy	This element should contain the URL that gives a direct link to the digital object.
europeana:provider	This element contains the name of the organisation that delivers data directly to Europeana.
dc:subject	The subject of the digital object which can include topics, people and places.
dc:type	The nature or genre of the digital object.
dc:coverage	Coverage can be used for either spatial or temporal aspects of the object being described.
dcterms: spatial	Information about the spatial characteristics of the digital object.
europeana:rights	The value in this element is a URL constructed by adding a code, indicating the copyright status of an object to the domain name where that status is defined.
europeana:type	This element is used to classify digital objects as one of the four Europeana material types: TEXT, IMAGE, SOUND or VIDEO.

Table 1: Mandatory Elements of the ESE format

Recommended Elements	
dcterms:alternative	This can be any alternative title or name by which the digital object is known.
dc:creator	The name of the creator or creators of the original physical object or the born digital object.
dc:contributor	The name of contributors to the either the original physical object or the born digital object.
dc:date	This date element should be used to contain the most significant date in the life of the digital object.
dcterms:created	This is the date of the creation of the digital object.
dcterms:issued	The date when the digital object was formally issued or published.
dcterms:temporal	Information of the temporal characteristics of the digital object.
dc:publisher	The name of the publisher of the digital object.
dc:source	This element should be used to indicate a related resource from which the digital object is derived.
dcterms:isPartOf	This element should be used to identify a related resource in which the described resource is physically or logically included.
europeana:object	This element supports the process of creating thumbnails for the Europeana portal.

Table 2: Recommended Elements of the ESE format

Elements supplied by Europeana	
europeana:country	This is the name of the country in which the organisation named in europeana:provider is based.
europeana:language	This is the official language of the country in which the organisation named in europeana:provider is located.
europeana:uri	The value in this element is a unique identifier for each object record in the Europeana system.
europeana:usertag	This element is provided to support future functionality in Europeana. These are tags created by registered users.
europeana:year	The value in this element is a four digit year (YYYY) from the Gregorian calendar used to support the Timeline and the Date facet in the portal.

Table 3: Europeana supplied elements of the ESE format

Additional Elements	
dc:format	This element can include the file format, physical medium and dimensions of the original physical object or the digital object.
dcterms:extent	Used to record the size or duration of the original physical or digital object.
dcterms:medium	The material or physical carrier of the resource.
dc:identifier	This element can be used for an identifier of the digital object.
dc:rights	Information about intellectual property rights, access rights or license arrangements for the digital object.
dcterms:provenance	This element is to record a statement of any changes in ownership and custody of the resource since its creation that are significant for its authenticity, integrity and interpretation.
dc:relation	This element should be used for information about resources that are related to the digital object.
dcterms:conformsTo	This element is used to identify standards to which the described resource conforms.
dcterms:hasFormat	This element used to identify another resource that is substantially the same as the digital object being described by the metadata but exists in a different format.
dcterms:isFormatOf	This element is used to identify a related resource that is substantially the same as the digital object but in a different format.
dcterms: hasVersion	This element is used to identify a related resource that is a version, edition or adaptation of the digital object described in the metadata.
dcterms:isVersionOf	This element is used to identify a related resource of which the described resource is a version, edition, or adaptation.
dcterms:hasPart	This element is used to identify a related resource that is included either physically or logically in the digital object.
dcterms:isReferencedBy	This element is used to identify a related resource that references, cites, or otherwise points to the digital object.
dcterms:references	This element is used to identify related resources that are referenced, cited, or otherwise pointed to by the digital object.
dcterms:isReplacedBy	This element is used to identify a related resource that supplants, displaces, or supersedes the digital object.
dcterms:replaces	This element is used to identify a related resource that is supplanted, displaced, or superseded by the digital object.
dcterms:isRequiredBy	This element is used to identify a related resource that requires the digital object to support its function, delivery or coherence.
dcterms:requires	This element is used to identify a related resource that is required by the digital object to support its function, delivery or coherence.

dcterms:tableOfContents	Used for a list of the sub-units of the digital object.
europeana:unstored	This element has been created in order to allow providers to retain all important information that cannot otherswise be mapped to ESE.

Table 4: Additional elements of the ESE format

2.2. Asynchronous JavaScript and XML (AJAX)

In 1990's user interaction in web applications was request-wait-response based, which slowed down the user interaction considerably. The most web sites were based on complete HTML pages where each user action required that the page should be re-loaded from the server. Each time a page was reloaded due to a partial change, all of the content was re-sent instead of only the changed information. This placed additional load on the server and use of excessive bandwidth. Asynchronous JavaScript and XML (AJAX) [5] came as a boon to the web application development, providing mechanisms for user experience similar to desktop applications.

In the classic web application model, addressed as pre AJAX web model, user interaction triggers an HTTP [6] request to the web server. The server performs necessary processing for example, retrieving data or doing some calculations etc. When the processing is completed the server returns an HTML [7] page to the client. The problem is that, during the server processing time, the user can do nothing but wait for a page to be loaded or refreshed from the server.

AJAX increases the web page's interactivity, speed, and usability in order to provide richer user experience. AJAX places an AJAX engine between the client and server. This engine is written in JavaScript and behaves like a hidden frame. The AJAX engine renders the user interface and handles the communication between client and server. The client-server communication with AJAX is asynchronous. Asynchronous communication means the client does not need to wait for the server response. After sending the request to the server the execution in the client program does not halt, rather the execution is continued. The response is sent to the client when it is available. The AJAX engine sends requests to the server on behalf of the client and receives data or responses from the server. In a web model with AJAX, the server sends small data instead of the HTML page. The AJAX engine shows the received data or response by updating the page partially. Thus user is free to do other interactions after sending a request to the server.

2.3. Java Servlet

A **Servlet** is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. The Servlet class is included in JAVA EE (Enterprise Edition) and conforms to the Java Servlet API, a protocol by which a Java class may respond to requests. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. Thus, it can be thought of as a Java Applet that runs on a server instead of a browser.

Most Java servlets are designed to respond to HTTP requests in the context of a Web application. As such, the HTTP-specific classes in the *javax.servlet* and *javax.servlet.http* packages are the ones you'll care about.

Each Java servlet is a subclass of *HttpServlet*. This class has methods that provide access to the request and response wrappers used to handle requests and create responses.

The HTTP protocol isn't Java-specific, of course. It is simply a specification that defines what service requests and responses have to look like. The Java servlet classes wrap those low-level constructs in Java classes, with convenience methods that make them easier to be used within a Java language context. When a user issues a request via a URL, the Java servlet classes convert it to an *HttpServletRequest* and send it to the target pointed to by the URL, as defined in configuration files for the particular servlet container the user is using. When the server side has done its work, the Java Runtime Environment packages the results in an *HttpServletResponse* and then sends a raw HTTP response back to the client that made the request. When a user is interacting with a Web app, he usually makes multiple requests and gets multiple responses. All of them are within the context of a session, which the Java language wraps in an *HttpSession* object.

A container, like Tomcat [8], manages the runtime environment for servlets. The container can be configured to customize the way in which the J2EE server functions. Various configurations allow the creation of a bridge from a URL (entered by a user in a browser) to the server-side components that handle the requests. When a web application starts, the container loads and initializes your servlet(s), and manages their lifecycle.

By the concept “servlet lifecycle”, we simply mean that things happen in a predictable way when a servlet is invoked. In other words, certain methods on any servlet will always get called in the same order. Here's a typical scenario:

- A user enters a URL in his browser. The Web server configuration file says that this URL points to a servlet managed by a servlet container running on the server.
- If an instance of the servlet hasn't been created yet (there's only one instance of a servlet for an application), the container loads the class and instantiates it.
- The container calls *init()* on the servlet.
- The container calls *service()* on the servlet, and passes in a wrapped *HttpServletRequest* and *HttpServletResponse*.
- The servlet typically accesses elements in the request, delegates to other server-side classes to perform the requested service and to access resources like databases, then populates the response using that information
- If necessary, when the servlet's useful life is done, the container calls *destroy()* on the servlet to finalize it.

2.4. XML

It is common for XML (Extensible Markup Language) [9] to be used in interchanging data over the Internet. XML is a markup language for documents containing structured information and was designed for data encoding and delivery. A markup language is a mechanism to specify structures in a document. The XML specification defines a standard way to add markup to documents.

XML is a flexible markup language that programmers can modify to cover their needs. That is, the programmer decides the XML tags that describe data rather than having to adhere to a standard set of tags as he does with HTML. It is worthwhile to mention that XML describes the data itself in contrast to HTML that describes how data should look on the screen. This flexibility allows the companies to create their own standard tags to describe data that is particular to their business.

XML is the most preferred way to transfer data between web applications, web services and systems based on web interaction for a couple of reasons. XML represents data, as a consequence, the data can be easily shared among different kinds of applications that run on different operating systems as it is independent from operating systems, transfer protocols and organizations and it is compatible with the majority of telecommunication protocols. Moreover, XML can be compressed in high levels in order to achieve faster transmission via networks.

2.4.1. XML Document, Elements and Attributes

An XML document contains nested elements; this way the relations between the tags of the XML document are implied. This hierarchy creates a tree of element nodes that depicts the data.

Every single element is enclosed in the angled brackets structural delimiters (< >), and always have an open (<) and closed markup tag (</). Child elements are placed within the open and closed markup tags of a parent element, and information is placed within the open and closed markup tags of a child element. The content type of an element may be plain text, a set of elements or combination of text and elements. An XML document must have one and only one root element, which includes the remaining elements of the data representation. Improperly nesting elements and orphan closing tags form an invalid xml document. A comment is information in the XML document that’s typically not part of the actual data and is enclosed in the (<!--) and (- ->) constructs.

Element tags may include attributes. An attribute is information that modifies an XML markup tag. Attributes are placed within the opening markup tag. One may create as many attributes as required; however, each attribute must have a unique name and a value contained within quotations (attributeName=“value”). Each name/value pair must be separated by a space. XML allows the flexibility to create custom attributes.

Element names and attribute names cannot contain whitespace characters; on the contrary, their values can contain whitespace.

```
<cellar>
        <wine>
                <title>Vinsanto</title>
                <producer country="Greece">
                        <Brandname>Santo Wines</Brandname>
                 </producer>
                <vintage>2003</vintage>
                 <!-- Vintage is the year the grapes were harvested. -->
                <purchase-date>July 2009</purchase-date>
        </wine>
</cellar>
```

Figure 1: XML document example

An example of an XML document is shown in Figure 1. This is a part of an XML document that carries information about a cellar. The hierarchy of "Cellar" and "wine" elements implies their "parent-child" relation. The "wine" element describes a bottle of wine in an individual's cellar and contains four elements: <title>, <producer>, <vintage>, <purchase-date> and one comment: <!-- Vintage is the year the grapes were harvested. -->. All elements have some text which is the information that is carried by the "wine" element. Furthermore, the <producer> element, except from its value, "SantoWines", contains an attribute named "country" that describes the origin of the wine producer.

2.4.2. Well-formed and valid documents

A **well-formed** XML document has correct XML syntax, which means that it follows the basic structural rules of XML:

- It contains one or more elements.
- There is exactly one element, the root or document element, no part of which appears in the content of any other element.
- The elements, delimited by start- and end-tags, nest properly within each other.
- The attribute values must be quoted.

Well-formed documents are well-formed because they do not have to be created in a structured environment, against a pre-defined set of structural rules, but merely have to comply with XML well-formedness constraints as presented above.

A **valid** XML document is a well-formed XML document, which also conforms to the rules of a Document Type Definition (DTD) or an XML Schema (XSD).

2.5. XML Schema

XML (Extensible Markup Language), described in Section 2.4, allows the developers to create their own formats for storing and sharing information. XML Schema [10] is the formal declaration and documentation of those formats, providing a foundation on which software developers can build software. XML Schema is the language for defining classes in XML documents. A set of attributes and content structure elements are defined for each class and its instances. An XML Schema language is a formalization of the constraints, expressed as rules or a model of structure, that apply to a class of XML documents. In many ways, schemas serve as design tools, establishing a framework on which implementations can be built.

The XML schema is expressed in XML 1.0 syntax and is intended to describe the structure and express the content constraints of documents written in XML. Thus, an XML Schema defines the structure of XML instance documents. The XML Schema comprises a set of components, which are categorized in three main groups: primary, secondary and helper components.

Like all XML schema languages, XSD can be used to express a set of rules to which an XML document must conform in order to be considered 'valid' according to that schema. However, unlike most other schema languages, XSD was also designed with the intent that determination of a document's validity would produce a collection of information adhering to specific data types.

2.5.1. Schema components

The **primary components** include Simple Type definitions, Complex Type definitions, Element definitions and Attribute definitions. Simple Types and Complex Types define the xml schema author-defined types that can be used across an xml schema by referring to them through the “type” attribute of elements. Complex Types may be defined as concrete or abstract, respectively allowing the derivation or not of instances, through the Boolean value of the optional attribute “abstract”. Simple Types are defined as restrictions, unions or lists of built-in types (like string, integer, double, Boolean etc.) or author-defined types of an xml schema. All kinds of types may or may not have a name, according to their location in the schema. Unnamed types are nested in other xml structures (elements, simple types etc.) and their scope is restricted within those structures. On the other hand, the element and attribute definitions must have a name.

Moreover, XML Schema supports mechanisms of inheritance both for types and elements, through the extension and restriction for the former and through the “substitutionGroup” for the latter. The types may include in their definition the “final” attribute with acceptable values “extension” and “restriction”, which prevents further derivations of a type by extension and restriction respectively. The SubstitutionGroup attribute supports the substitution of one and only named element from another. Any top-level element declaration can serve as the defining member, or head, for an element substitution group.

Other top-level element declarations, regardless of the target namespace, can be designated as members of the substitution group headed by this element. Note that the members must have type definitions which are either the same as the head's type definition or restrictions or extensions of it.

The **secondary components** include attribute group definitions, identity constraint definitions, model group definitions and notation declarations. The attributes capture information about complex types and form attribute groups when there is need to use simultaneously all of them. Attribute groups must be declared in top-level, in contrary to individually declared attributes that may appear at top-level or nested in complex types. The same fact is applied to model groups, too. Model groups are collections of elements with a specific behavior among them. A model group specifies a sequential (sequence), disjunctive (choice) or conjunctive (all) interpretation of the group members. The elements taking part in sequences, choices and all structures, may carry the “minOccurs” or/and the “maxOccurs” attribute in their declaration. These attributes specify, respectively, the minimum and the maximum number of occurrences of each element in the model group. The default value, in case of absence, is 1, implying exactly one occurrence.

The **helper components** include annotations, model groups, particles (i.e. min occurs and max occurs), wildcards and attribute uses. These components essentially are small parts of other components, since they are not independent of their context.

2.5.2. Identity constraints

The identity constraint mechanism of XML Schema is a very powerful tool for schema authors. It basically provides two ways to identify and reference elements and attributes inside an XML document. These are the ID/IDREF/IDREFS and key/keyref. The former combination is inherited from XML’s DTDs while the latter is introduced by the XML Schema in order to offer more flexibility by using XPath. The usage semantics of the identity constraints are close to those of foreign keys in relational databases.

The first approach to describe identifiers and references with W3C XML Schema, ID/IDREF, can be used to define either attributes or elements by binding their type to the xs:ID, xs:IDREF or xs:IDREFS built-in datatypes. IDREFS is a whitespace-seperated list of IDREF values. IDs are global to a document, meaning that we are not allowed to use the same ID value for different constructs within the same document and indicates that the attribute value uniquely identifies the containing element. In Figure 2 and Figure 3 are presented an XML Schema that uses ID/IDREF/IDREFS and a sample XML document based on it, respectively.

```
<xs:element name="wine">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="title" type="xs:string" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="producer" minOccurs="1" maxOccurs="1">
                                <xs:complexType>
                                        <xs:attribute name="country" use="required" type="xs:string"/>
                                        <xs:attribute name="ref" type="xs:IDREF" use="required"/>
                                </xs:complexType>
                        </xs:element>
                        <xs:element name="vintage" type="xs:IDREFS" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="purchase-date" type="xs:string" minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
                <xs:attribute name="wineID" type="xs:ID" use="required"/>
        </xs:complexType>
</xs:element>
```

Figure 2 : XML Schema Element with ID/IDRef/IDRefs

This is a possible xml schema that describes a wine bottle in a cellar. Notice that the attributes values of the ID and IDREF types cannot start with a number. Also, notice that element “producer” cannot have any children, since the IDREF content type is empty. Moreover, the element “vintage” is a whitespace-separated list of years. Finally, the years start with the “Y” letter, because an IDREF value cannot start with a number. Under these limitations, a sample XML document based on the schema presented in Figure 2 is shown in Figure 3.

```
<wine wineID="IDWINE021">
        <title>Vinsanto</title>
        <producer country="Greece" ref="IDPROD0036"/>
        <vintage>Y2003 Y2005</vintage>
        <purchase-date>07-2009</purchase-date>
</wine>
```

Figure 3 : XML document with ID/IDRef/IDRefs

The identity constraints of the key/keyref type contain: (a) A selector element, which specifies the XML Schema elements on which the identity constraint is applied; and (b) A field element, where the XML Schema constructs (elements or attributes) that form the constraint value are specified. Both the selector and field elements specify the construct(s) they refer to in their xpath attribute. The value of that attribute is a restricted XPath (XPath 1.0 [W3C/XSD1]) expression reffering to instances of the element being declared. The enhancements of the ID/IDREF mechanism are the following: a)Functioning as a part of an identity-constraint is in addition to, not instead of, having a type; b)Not only attribute values, but also element content and combinations of values and content can be declared to be unique; c)Identity-constraints are specified to hold within the scope of particular elements; d)(Combinations of) attribute values and/or element content can be declared to be keys, that is, they are not only unique, but always present and non-nillable; and e) The comparison between keyref {fields} and key or unique {fields} is by value equality, not by string equality.

```
<xs:element name="wine">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="title" type="xs:string" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="producer" minOccurs="1" maxOccurs="1">
                                <xs:complexType>
                                        <xs:all>
                                                <xs:element name="Brandname" type="xs:string"/>
                                        </xs:all>
                                        <xs:attribute name="country" use="required" type="xs:string"/>
                                </xs:complexType>
                        </xs:element>
                        <xs:element name="vintage" type="xs:integer" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="purchase-date" type="xs:string" minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
        </xs:complexType>
        <xs:unique name="UniqueBottle">
                <xs:selector xpath="wine"/>
                <xs:field xpath="@title"/>
                <xs:field xpath="producer/Brandname"/>
                <xs:field xpath="vintage"/>
        </xs:unique>
</xs:element>
```

Figure 4 : XML Schema element with Unique identity constraint

2.5.3. Namespaces

The Namespaces provide a URI-based mechanism that allows differentiating XML vocabularies. XML Schema associates a namespace to all the objects (elements and attributes, but also simple and complex types as well as groups of elements and attributes) defined in a schema, allowing the use of namespaces to build modular libraries of schemas. A Namespace is a URI represented by a prefix. The default XML Schema namespace, which includes built in types is "http://www.w3.org/2001/XMLSchema" and is often represented by the "xs" prefix. Namespace prefixes should only be considered to be local shortcuts to replace the URI references that are the real identifiers for a namespace.

2.6. XML Beans

XMLBeans [11] is a tool that allows access to the full power of XML in a Java friendly way. The idea is that you can take advantage of the richness and features of XML and XML Schema and have these features mapped as naturally as possible to the equivalent Java language and typing constructs. XMLBeans uses XML Schema to compile Java interfaces and classes that can be used to access and modify XML instance data. While a major use of XMLBeans is to access an XML instance data with strongly typed Java classes there are also API's that allow access to the full XML Infoset (XMLBeans keeps XML Infoset fidelity) as well as to allow reflection into the XML schema itself through an XML Schema Object model.

There are at least two major things that make XMLBeans unique from other XML-Java binding options.

1. **Full XML Schema support.** XMLBeans fully supports XML Schema and the corresponding java classes provide constructs for all of the major functionality of XML Schema. This is critical since many times there isn't control over the features of XML Schema that are needed to be exploited in Java. Also, XML Schema oriented applications can take full advantage of the power of XML Schema and they do not have to restrict themselves to a subset.
2. **Full XML Infoset fidelity.** When unmarshalling an XML instance (converting XML instance into java object) the full XML infoset is kept and is available to the developer. This is critical because of the subset of XML that is not easily represented in java. For example, the order of the elements or comments might be needed in a particular application.

A major objective of XMLBeans has been to be applicable in all non-streaming (in memory) XML programming situations. You should be able to compile your XML Schema into a set of java classes and know that: a) you will be able to use XMLBeans for all of the schemas you encounter (even the warped ones) and b) that you will be able to get to the XML at whatever level is necessary - and not have to resort to multiple tools to do this.

To accomplish this XMLBeans provides three major APIs:

- **XmlObject:** The java classes that are generated from an XML Schema are all derived from XmlObject. These provide strongly typed getters and setters for each of the elements within the defined XML. Complex types are in turn XmlObjects. For example getCustomer might return a CustomerType (which is an XmlObject). Simple types turn into simple getters and setters with the correct java type. For example getName might return a String.
- **XmlCursor**: From any XmlObject you can get an XmlCursor. This provides efficient, low level access to the XML Infoset. A cursor represents a position in the XML instance. You can move the cursor around the XML instance at any level of granularity you need from individual characters to Tokens.
- **SchemaType**: XMLBeans provides a full XML Schema object model that you can use to reflect on the underlying schema meta information. For example, you might want to generate a sample XML instance for an XML schema or perhaps find the enumerations for an element so that you can display them.

2.7. Apache Lucene/Solr

Apache Lucene [12] is a free/open source information retrieval software library, originally created by Doug Cutting. It is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.

While suitable for any application which requires full text indexing and searching capability, Lucene has been widely recognized for its utility in the implementation of Internet search engines and local, single-site searching.

At the core of Lucene's logical architecture is the idea of a document containing fields of text. This flexibility allows Lucene's API to be independent of the file format. Text from PDFs, HTML, Microsoft Word, and OpenDocument documents, as well as many others, can all be indexed as long as their textual information can be extracted.

Solr [13] is the popular, blazing fast open source enterprise search platform from the Apache Lucene project. Its major features include powerful full-text search, hit highlighting, faceted search, dynamic clustering, database integration, rich document (e.g., Word, PDF) handling, and geospatial search. Solr is highly scalable, providing distributed search and index replication, and it powers the search and navigation features of many of the world's largest internet sites.

Solr is written in Java and runs as a standalone full-text search server within a servlet container such as Tomcat [8]. Solr uses the Lucene Java search library at its core for full-text indexing and search, and has REST-like HTTP/XML and JSON APIs that make it easy to use from virtually any programming language. Solr's powerful external configuration allows it to be tailored to almost any type of application without Java coding, and it provides extensive plugin architecture when more advanced customization is required.

2.8. Resource Description Framework (RDF)

RDF (Resource Description Framework) [14] is actually one of the older specifications, with the first working draft produced in 1997. In the earliest version, authors established a mechanism for working with metadata that promotes the interchange of data between automated processes. This mechanism became the base on which RDF was developed. Regardless of the transformations RDF has undergone and its continuing maturing process, this statement forms its immutable purpose and focal point.

The Resource Description Framework (RDF) is a language designed to support the Semantic Web, in much the same way that HTML is the language that helped initiate the original Web. RDF is a framework for supporting resource description, or metadata (data about data), for the Web. RDF provides common structures that can be used for interoperable XML data exchange.

One of the differences between XML and RDF is about the tree-structured nature of XML, as compared to the much flatter triple-based pattern of RDF. XML is hierarchical, which means that all related elements must be nested within the elements they are related to. RDF does not require this nested structure. On the other hand, XML does not provide any information about the data described. That is, the nesting structure of the nodes does not imply in any way the relations among the data described, but only which element is the parent of the other element. In contrast to this fact, an RDF triple pattern gives the information how a datum is related to the rest of the data of the domain.

2.8.1. Basic features of RDF

RDF conceptualizes anything (and everything) in the universe as a **resource**. A resource is simply anything that can be identified by a Universal Resource Identifier (URI). URI provides a unique identifier for the information. Anything whether we can retrieve it electronically or not, can be uniquely identified in a similar way.

An **RDF triple** is formed by three components (predicate, subject, and object). These components create a **statement**, subject – predicate – object, in which the predicate specifies the relation between the subject and the object. The subject and the object may be any resources or literals (literals are atomic values, strings). Moreover, predicates are also known as **properties**. RDF properties may be thought of as attributes of resources and in this sense they correspond to traditional attribute-value pairs. RDF properties represent relations between resources. Let P be a predicate and x, y the subject and the object respectively (x, P, y).The property P can be regarded as a logical function: P(x,y) of two inputs, which describes the relation between x and y.

Furthermore, RDF triples can be described by an RDF graph, a directed labeled graph that contains nodes and arcs. The RDF triple is comprised of a resource node (the subject) which is linked to another resource node (the object) through an arc labeled with a third resource (the predicate). In Figure 5 is shown an RDF graph that contains one triple.

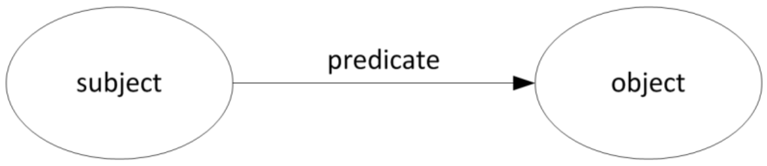


Figure 5 : The RDF graph of a RDF triple

Unfortunately, recording the RDF data in a graph is not the most efficient means of storing or retrieving this data. Instead, encoding RDF data, a process known as serialization, is usually preferred, resulting in *RDF/XML* which is based on XML structures. An RDF/XML document starts with the root element *rdf:RDF*. The children of this element include the data descriptions (*rdf:Description*). A description element expresses a statement about one resource. The reference to that resource can be established by a) Using the *rdf:id*, in case the resource is new; b) Using the *rdf:about*, in case to refer to an existing resource; or c) Without using any name in order to be anonymous. Anonymous resources belong to a special category of nodes, the blank nodes, defined in Definition 2.1.

Blank nodes are nodes that don't have a URI. When identifying a resource is meaningful, or the resource is identified within the specific graph, a URI is given for that resource. However, when the identification of the resource does not have sense or does not exist within the specific graph at the time the graph was recorded, the resource is treated as a blank node. In either case, not having a URI for the item does not mean we cannot talk about it and refer to it. Blank nodes are graph nodes that represent a subject (or object) for which we would like to make assertions, but have no way to address with a proper URI.

The following RDF graph describes that "*John has a friend born the 26th of January*". This expression can be written with two triples linked by a blank node representing the anonymous friend of John.

ex:John	*foaf:knows*	*_:p1*
_:p1	*foaf:birthDate*	*01-26*

The first triple specifies that "*John knows p1*". The second triple specifies that "*p1 is born on January 26th*". Moreover, "*ex:John*" is a named resource, which means this resource is absolutely identified by the URI obtained by replacing the "*ex:*" prefix by the XML namespace it stands for, such as http://music.tuc.gr/Person#John. The "*_:p1*" blank node represents John's anonymous friend, not identified by a URI. One can know by the semantics declared in the FOAF vocabulary [FOAF] that the class of "*_:p1*" is "*foaf:Person*".

2.9. Simple Knowledge Organization System (SKOS)

Simple Knowledge Organization System (SKOS) [15] is a family of formal languages designed for representation of thesauri, classification schemes, taxonomies, subject-heading systems, or any other type of structured controlled vocabulary. Using SKOS, a knowledge organization system can be expressed as *machine-readable data*. It can then be exchanged between computer applications and published in a machine-readable format in the Web. The SKOS data model is formally defined in this specification as an OWL Full ontology [16] [OWL-SEMANTICS]. SKOS data are expressed as RDF triples, and may be encoded using any concrete RDF syntax (such as RDF/XML [17] or Turtle [18]).

The SKOS data model views a knowledge organization system as a *concept scheme* comprising a set of *concepts*. These SKOS concept schemes and SKOS concepts are identified by URIs, enabling anyone to refer to them unambiguously from any context, and making them a part of the World Wide Web. SKOS concepts can be *labeled* with any number of lexical (UNICODE) strings in any given natural language. One of these labels in any given language can be indicated as the preferred label for that language, and the others as alternative labels. Labels may also be "hidden", which is useful where a knowledge organization system is being queried via a text index. SKOS concepts can be *linked* to other SKOS concepts via semantic relation properties. The SKOS data model provides support for hierarchical and associative links between SKOS concepts. Again, as with any part of the SKOS data model, these can be extended by third parties to provide support for more specific needs. SKOS concepts can be grouped into **collections**, which can be labeled and/or ordered. This feature of the SKOS data model is intended to provide support for node labels within thesauri, and for situations where the ordering of a set of concepts is meaningful or provides some useful information. Finally, SKOS concepts can be **mapped** to other SKOS concepts in different concept schemes. The SKOS data model provides support for four basic types of mapping link: hierarchical, associative, close equivalent and exact equivalent.

Figure 6 presents the SKOS representation of a concept found in the UK Archival Thesaurus (UKAT).

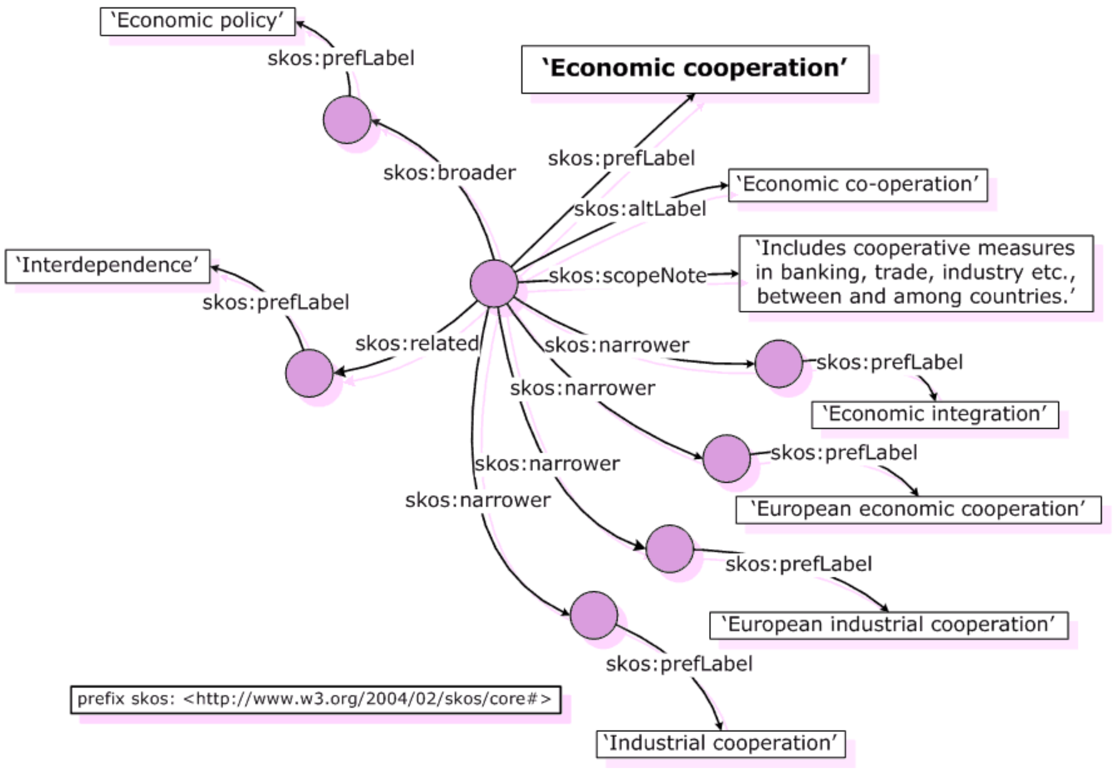


Figure 6: SKOS example

2.10. Google Web Toolkit (GWT)

Google Web Toolkit (GWT) [19], first released in May 2006, is an open source development toolkit for building and optimizing complex browser-based applications. Its goal is to enable productive development of high-performance web applications without the developer having to be an expert in browser quirks.

Today's RIAs (Rich Internet Applications, i.e. desktop like web applications) are getting complex and large in size. Writing AJAX applications in JavaScript is error prone. Managing large applications in JavaScript is complex, difficult and directs to an entirely new discipline. Also, JavaScript behaves differently on different browsers. Developers spend a lot of valuable time having to code for browser differences instead of focusing on real application logic. Also, developers tend to mix the business logic in the view of web applications using JavaScript.

RIA application development with GWT gives Java developers the ability to reuse their existing expertise and best practices. GWT gives ease of developing large applications, as Java was designed to make large applications manageable in object oriented fashion. With GWT, besides, having all the advantages of Java as a programming language, developers can use a large number of Java development tools that already exist. They can use their favorite IDE, perform compile time checking, unit testing and even continuous integration. GWT also handles all browser-specific quirks meaning that compiled GWT application runs inside any modern browser (assuming JavaScript is turned on), so that developers can focus on the application logic. GWT basically translates all the Java UI code to JavaScript. However, it does not mean that the old JavaScript code or application will become useless. GWT still allows interacting with existing JavaScript code as well as integrating with existing server side services. Another important functionality GWT provides is that, it separates server side logic from client side creating separate packages for the client and the server.

Google provides a plugin for Eclipse which handles most GWT related tasks in the IDE including creating projects, invoking the GWT compiler, creating GWT launch configurations, validations, syntax highlighting, etc. In our thesis work, we have used a GWT SDK of version 2.3 plugin to work in Eclipse environment.

2.10.1. GWT Components

GWT provides a comprehensive set of tools including UI components to configuration tools to server communication techniques and this help web applications look, act, and feel like full-featured desktop applications. Major GWT components are as follows.

- GWT Java-to-JavaScript Compiler
  This is the core part of GWT. This compiler converts Java code into JavaScript code in such a way, that the compiled JavaScript can run on the major internet browsers. The supported browsers include Internet Explorer, Firefox, Mozilla, Opera, and Safari.

- JRE emulation library
  To provide developer to use some classes of core Java, GWT includes JRE (Java Runtime Environment) emulation library. This library supports some classes from *java.lang* and *java.util* packages.

- GWT Web UI class library
  GWT ships with a large set of custom interfaces and classes for creating widgets and panels. Widget is some sort of control used by a user, and a panel is a container into which controls can be placed.

2.10.2. GWT Modes of Running

GWT applications can run on two following modes.

- Development Mode (Hosted Mode)
  This mode can be addressed as debug mode also. With this mode, GWT allows developers to debug their application as in any Java application. In this mode, Java code is executed and widgets are displayed in a host window that emulates a web browser.

- Web Mode
  This mode executes the JavaScript code generated from the compilation of client side Java code. The Web mode is actually the deployment of Ajax web application with GWT framework on a genuine application server.

2.10.3. Remote Procedure Calls (RPCs)

GWT provides an RPC mechanism based on Java Servlets to provide access to server side resources. This mechanism includes generation of efficient client and server side code to serialize objects across the network using deferred binding.

A fundamental difference between AJAX applications and traditional HTML web applications is that AJAX applications do not need to fetch new HTML pages while they execute. Because AJAX pages actually run more like applications within the browser, there is no need to request new HTML from the server to make user interface updates. However, like all client/server applications, AJAX applications usually *do* need to fetch data from the server as they execute. The mechanism for interacting with a server across a network is called making a remote procedure call (RPC). GWT RPC makes it easy for the client and server to pass Java objects back and forth over HTTP.

The server-side code that gets invoked from the client is often referred to as a ***service***, so the act of making a remote procedure call is sometimes referred to as invoking a service.

Java components of the GWT RPC Mechanism

When setting up GWT RPC, we focused on these three elements involved in calling procedures running on remote servers.

- the service that runs on the server (the method we are calling)
- the client code that invokes the service
- the Java data objects that pass between the client and server.

Both the server and the client have the ability to serialize and deserialize data so the data objects can be passed between them as ordinary text.

In Figure 7, they are depicted the components of the GWT RPC Mechanism.

As we can see, in order to define our RPC interfaces (in this example, CollectionAccessService interface), we need to:

1. Define an interface (CollectionAccessService) for our service that extends *RemoteService* interface and lists all our RPC methods.
2. Define a class (CollectionAccessServiceImpl) to implement the server-side code that extends *RemoteServiceServlet* and implements the interface we created above.
3. Define an asynchronous interface (CollectionAccessServiceAsync) to our service to be called from the client-side code.

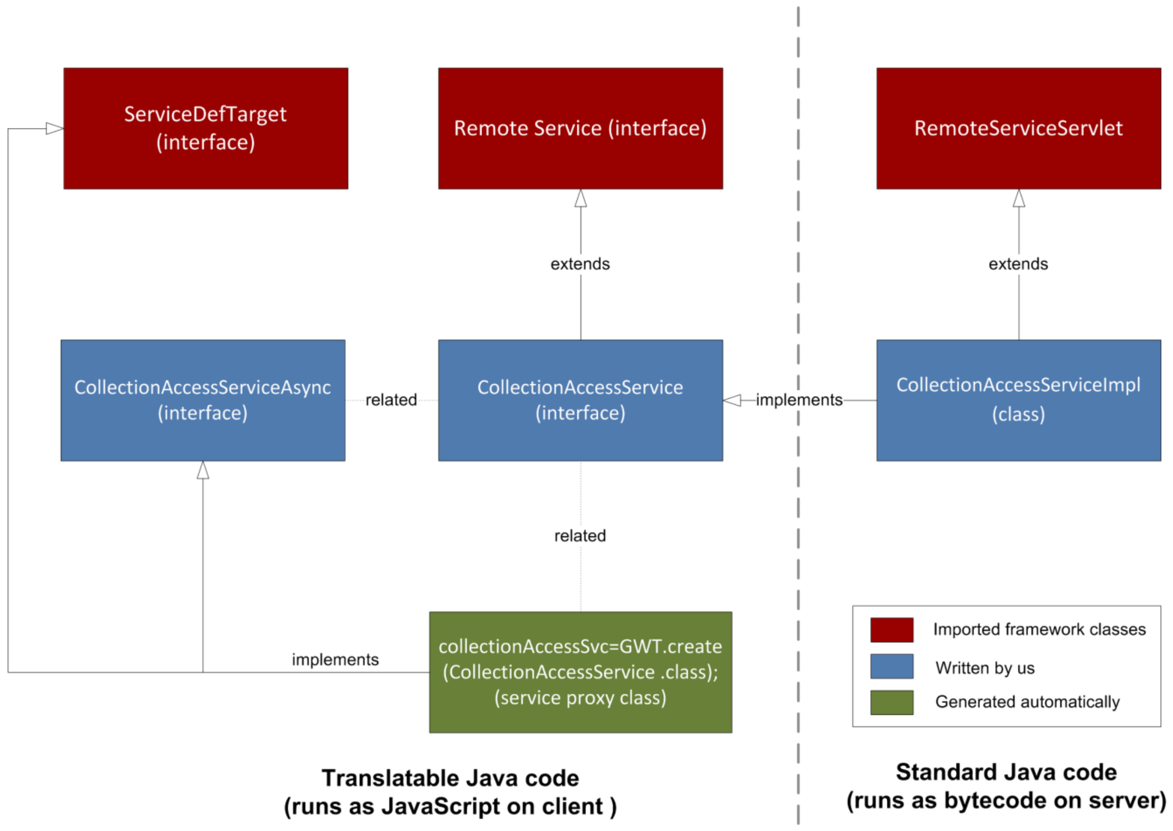

ServiceDefTarget (interface)
Remote Service (interface)
RemoteServiceServlet
extends
extends
CollectionAccessServiceAsync (interface)
related
CollectionAccessService (interface)
implements
CollectionAccessServiceImpl (class)
related
implements
collectionAccessSvc=GWT.create (CollectionAccessService .class); (service proxy class)
Imported framework classes
Written by us
Generated automatically
Translatable Java code (runs as JavaScript on client)
Standard Java code (runs as bytecode on server)

Figure 7: Components of GWT RPC Mechanism

The nature of asynchronous method calls requires the caller to pass in a callback object that can be notified when an asynchronous call completes, since by definition the caller cannot be blocked until the call completes. For the same reason, asynchronous methods do not have return types; they generally return void. After an asynchronous call is made, all communication back to the caller is via the passed-in callback object.

The name convention that is adopted for our services is:

- The interface for each service that lists all its RPC methods is named as *ServiceName***Service.**
- The class that implements the server-side code and implements the previous interface is named as *ServiceName***ServiceImpl.**

  Finally, the asynchronous interface to the service that is called from client-side is named as *ServiceName***ServiceAsync.**

2.10.4. Locales in GWT

GWT is different than most toolkits by performing most locale-related work at compile time rather than runtime. This allows GWT to do compile-time error checking, such as when a parameter is left out or the translated value is not of the correct type, and for optimizations to take into account known facts about the locale. This also allows an end user to download only the translations that are relevant for them.

Internationalization Techniques

GWT offers multiple internationalization techniques to afford maximum flexibility to GWT developers and to make it possible to design for efficiency, maintainability, flexibility, and interoperability in whichever combinations are most useful.

- **Static string internationalization**

  Static string internationalization requires very little overhead at runtime and therefore is a very efficient technique for translating both constant and parameterized strings. It is also the simplest technique to implement. Static string internationalization uses standard Java properties files to store translated strings and parameterized messages, then implements strongly-typed Java interfaces to retrieve their values.

- **Dynamic string internationalization**

  Dynamic string internationalization is slower than static string internationalization, but is very flexible. Applications using this technique look up localized strings in the module's host page; therefore, they do not need to be recompiled when it is added a new locale. If it is needed to integrate a GWT application with an existing server-side localization system, dynamic string internationalization is the option to consider.

- **Extending or implementing Localizable**

  The most powerful technique is to implement the Localizable interface. Implementing Localizable allows you to go beyond simple string substitution and create localized versions of custom types. It's an advanced internationalization technique that you probably won't have to use very often.

Resource Bundles

In standard Java programming, internationalization is usually done by means of resource bundles: .properties files with locale-specific data. Although this data might be anything (numbers, dates, whatever) we have dealt only with strings. Each string is identified by a “key,” which must remain constant across different resource bundles. Basically, in our code we are referring to this key so our program will be locale-independent, inasmuch as what string will be shown shall depend on which locale resource bundle we use. GWT supports generic resource bundles (strings that will be shown if no other more specific locale is chosen), language resource bundles (for example, English or Spanish versions of our strings), and even country-specific resource bundles (such as British English, or Mexican Spanish). In our case, we have a generic bundle file, plus two language bundles (one for English and one for Greek). All keys appear in the generic bundle file. If a certain key appears in several bundles, language strings have priority over the generic ones. An example of bundles can be seen in Table 5.

Transport.properties	**Transport_en.properties**	**Transport_el.properties**
flight=airplane	flight=aeroplane	flight=αεροπλάνο
Vehicle=car		vehicle=αυτοκίνητο
underground=subway	underground=tube	underground=μετρό

Table 5: Internationalization Example in GWT.

In this case, a British user who wanted to use the underground would get a message about the *tube*; Greek users would get references to the *μετρό*; everybody else (including other non-British English speakers) would get the *subway* standard reference. (We consider English as the standard language.)

Resource bundles are named with the interface name, followed by an underscore and a lowercase two-character language specification. For example, the resource bundle for the constants interface in Greek is named as **Constants_el.properties** file. The two-character language specification was taken from ISO 639-1 list. ISO 639 is a standardized nomenclature used to classify all known languages. Each language is assigned a 2-letter (639-1) lowercase abbreviation, amended in later versions of the nomenclature.

Finally, the resource bundles were written in UTF-8 encoding in order to represent every character in the Unicode character set.

Constants

The Constants interface allows you to localize constant values in a type-safe manner, all resolved at compile time.

Constants interface is used in order to create a collection of constant values of a variety of types that can be accessed by calling methods (called *constant accessors*) on an interface. Constant accessors may return a variety of types, including strings, numbers, booleans, and even maps. A compile-time check is done to ensure that the value in a properties file matches the return type declared by its corresponding constant accessor. In other words, if a constant accessor is declared to return an integer, its associated property is guaranteed to be a valid integer value — avoiding a potential source of runtime errors.

The Constants interface bind, at compile time, the provided resource bundles with our provided code, to produce locale-specific versions of the code. Whenever the user browses to our application, the loader code determines user’s browser type and his locale, and then loads the compiled version of our system that matches those two parameters.

Messages

The Messages interface is used so as to create a collection of formatted messages that can accept parameters. It substitutes parameters into messages and even re-orders those parameters for different locales as needed. The format of the messages in the properties files follows the specification in Java MessageFormat. The interface it is created contains a Java method with parameters matching those specified in the format string. In addition, the Messages interface supports Plural Forms to allow application to accurately reflect text changes based on the count of something.

2.10.5. UIBinder framework

The UiBinder framework is used in order to build Widget and DOM structures from XML markup. It allows you to build apps as HTML pages with GWT widgets sprinkled throughout them.

Besides being a more natural and concise way to build a UI than doing it through code, UiBinder can also make an app more efficient. Browsers are better at building DOM structures by cramming big strings of HTML into innerHTML attributes than by a bunch of API calls.

The UiBinder framework:

- helps productivity and maintainability — it's easy to create UI from scratch or copy/paste across templates;
- makes it easier to collaborate with UI designers who are more comfortable with XML, HTML and CSS than Java source code;
- provides a gradual transition during development from HTML mocks to real, interactive UI;
- encourages a clean separation of the aesthetics of a UI (a declarative XML template) from its programmatic behavior (a Java class);
- performs thorough compile-time checking of cross-references from Java source to XML and vice-versa;
- offers direct support for internationalization that works well with GWT's i18n facility and
- encourages more efficient use of browser resources by making it convenient to use lightweight HTML elements rather than heavier-weight widgets and panels.

However uiBinder is not a renderer. There are no loops, no conditionals, no if statements in its markup. UiBinder allows you to lay out widgets. It's still up to the widgets themselves to convert rows of data into rows of HTML.

Here's a very simple example of a UiBinder template that contains no widgets, only HTML:

```xml
<!-- HelloWorld.ui.xml -->
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'>
  <div>
    Hello, <span ui:field='nameSpan'/>.
  </div>
</ui:UiBinder>
```

Below it is an example of a UiBinder template that uses the HorizontalPanel layout widget in order to align horizontally a label and a button widget.

```xml
<!-- HelloWidgetWorld.ui.xml -->
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'
  xmlns:g='urn:import:com.google.gwt.user.client.ui'>
  <g:HorizontalPanel>
     <g:Label>Click me :</g:Label>
     <g:Button>Save</g:Button>
  </g:HorizontalPanel>
</ui:UiBinder>
```

In order to use a set of widgets in a ui.xml template file, we needed to tie their package to an XML namespace prefix. That's what's happening in this attribute of the root `<ui:uiBinder>` element: `xmlns:g='urn:import:com.google.gwt.user.client.ui'`. This says that every class in the `com.google.gwt.user.client.ui` package can be used as an element with prefix `g` and a tag name matching its Java class name, like `<g:Label>`.

Chapter 3

Functional Specification

This chapter describes the functional specification of the system that has to be deployed at each museum in order to complete the ingestion, maintenance, curation and dissemination of CHOs and metadata. It presents the definitions used throughout the document, specifies the stakeholders of the system, lists the technical requirements that have to be met for achieving the desired functionality and provides an in depth analysis for the tool's functionality.

To this end, Section 3.1 contains the definitions of the terms used in the whole document, Section 3.2 presents the system's stakeholders and their role in the NH Cultural Heritage Portal of the Natural Europe, while Section 3.3 discusses the technical requirements of the tool. Finally, Section 3.4 provide an in depth analysis of the functionality that has to be provided by the system in the form of use cases.

3.1. Definitions

- **Cultural Heritage Object (CHO):** refers to any type of cultural heritage content item that is digitized and that belongs to a collection held by a cultural institution. Such collections may belong to public cultural and scientific institutions (e.g. libraries, archives and museums) as well as private content holders (e.g. publishers). They may include all types of physical cultural items that have been digitized (e.g. books, audiovisual or multimedia material, photographs, documents in archives etc.) or material originally produced in digital format.
- **Cultural Heritage Object Collection (CHO Collection):** Any collection of Cultural Heritage Objects and/or Cultural Heritage Object Metadata.

- **Cultural Heritage Object Collection Metadata (CHO Collection Metadata):** A description about the common characteristics of the CHOs in the context of a CHO Collection that follows a specific structure (e.g., DC, etc.).
- **Cultural Heritage Object Metadata (CHO Metadata):** Description about a Cultural Heritage Object that follows a specific structure (e.g., ESE, DC, etc.).
- **Cultural Heritage Object Thumbnail (CHO Thumbnail):** A representative image for a published Cultural Heritage Object.
- **Published Cultural Heritage Object (Published CHO):** Any Cultural Heritage Object that can be accessed (in the original, or lower quality) by the end users through the Web and not necessarily for free.

3.2. Stakeholders

In Natural Europe the participating NHMs will contribute metadata about a large number of Natural History related CHOs which will be aggregated by the project and exploited for educational purposes. These CHO metadata will be further harvested by Europeana.eu in order to become available through its portal. Moreover, the NHMs will semantically enrich the contributed descriptions with Natural Europe shared knowledge (shared vocabularies, taxonomies, etc.).

In the perspective under consideration (as a Cultural Heritage Environment), the system stakeholders are:

- **The Natural History Museums.** The participating NHMs are the main content providers, contributing CHO metadata records that will be exploited, as a next step, for learning purposes. This means that appropriate descriptions and semantic enrichments have to be done. Before that, it must be ensured that any existing descriptions that NHMs have about their objects will be properly migrated, reused and exploited in the Natural Europe environment.
- **Natural Europe Federation.** The metadata records provided by the NHMs will be aggregated in the Natural Europe Federation. The aggregated content will then be available for dissemination to the Europeana.eu, as well as to third-party federations like the BHL [20] the Organic.Edunet [21], etc.
- **The Europeana.eu.** The contributed CHO metadata records of each participating NHM will be made available through the Europeana.eu Portal after being harvested from the Natural Europe Federation. Therefore each CHO has to be published, described, licenced, and maintained following the Europeana.eu specifications and policies.

3.3. Technical requirements

This section discusses the technical requirements that were set in the development of the EuroMuse tool, based on the Europeana.eu metadata submission practices, Europeana.eu proposed software tools, as well as basic NHM requirements.

To this end, the basic technical requirements that have to be met are summarized below:

- **Application Development:** The tool supporting the CHO metadata annotation in each NHM should be easy to install. For that, a web application delivered via a typical Web browser installed in a single NHM's server seems to be the best choice, compared to a desktop application that requires installation in multiple NHM's terminals. Moreover this choice seems to empower the collaboration between the NHM's curators that will be the main users of the system. To this end, Europeana.eu proposes the Google Web Toolkit (GWT) [19] for Web application development. GWT is a platform for developing Rich Internet Applications (RIA), which are typical web applications having many of the characteristics of desktop applications.
- **User Support:** EuroMuse has to support different user categories having specific access rights. This is a fundamental requirement in almost every content management application. At least the following three user categories have to be supported:
    - **Administrator:** Allowed to review/create/update/delete user accounts, CHO collections and CHO metadata records.
    - **Curator:** Allowed to review/create/update/delete CHO collections and CHO metadata records.
    - **Guest:** Allowed only to review CHO collections and CHO metadata records.
- **Conformance to ESE/EDM Metadata Models:** The application to be developed is required to support the metadata enrichment of a CHO using Europeana.eu compliant schemas (i.e., ESE [2] and EDM [22] schemas) since the contributed metadata records will be aggregated, in a later step, to the Europeana.eu.
- **Metadata Unification:** The migration of already existing NHM legacy metadata to ESE compliant records has to be also supported. Europeana.eu offers the SIP Creator tool [23], a standalone Java application (although web-launched) designed to support the easy transformation of any record-based XML input into the ESE format. Using this tool, the NHM curators will be able to transfer their legacy metadata to the NHM Cultural Environment tools, for further enrichment.
- **CHO Publishing:** The publishing of NHM CHOs is required to be supported, since the Europeana.eu receives metadata descriptions only for Web accessible resources.
- **Thumbnail Creation:** In case that the NHMs do not have published thumbnails for their CHOs, the tools should be able to automatically create and publish CHO thumbnails. This functionality is important to be supported, since the URL of the CHO thumbnail is included in the ESE metadata elements, and according to

Europeana.eu policies, the metadata records that do not contain a thumbnail URL for the described object will appear last in the search results of the Europeana.eu in the future. To this end, the use of ImageMagick [24] and FFmpeg [25] for thumbnail creation is proposed by the Europeana.eu. ImageMagick is an open source software suite for displaying, converting and editing raster image files, while FFmpeg is an open source project that produces libraries and programs for handling multimedia data including video files.

- **OAI-PMH Compliant Metadata Dissemination:** The repository of the application has to implement the OAI-PMH [26] interface, in order to enable the CHO metadata harvesting from the Natural Europe Federation. The OAI-PMH specifies a method for digital repositories ("data providers") to expose metadata about their objects for harvesting by aggregators ("service providers"). It is used by several organizations and Europeana.eu considers it as a best practice for metadata harvesting.

3.4. Use cases

EuroMuse is the front-end application that each museum will use for the semantic annotation of its CHOs, as well as for the publishing of the CHOs in the case that the contributed CHOs are not already published. This section presents an analysis of the basic functionality of the application in the form of use cases.

EuroMuse is basically used to facilitate the importing, organization, and describing of the museum's CHOs by the museum curators. Its main functionality includes the following operations:

- Create/delete/update/review CHO Collections
- Describe CHO Collections (with appropriate metadata)
- Create/delete/update/review/import/export CHO Metadata
- Import/publish/update/delete CHOs
- Create/delete/update/manage Users

The primary actors of the application are the users (i.e., physical persons) acting upon it to achieve certain goals (e.g., the import of a CHO in the system). According to the technical requirements described in Section 3.3, these users are grouped in three categories (minimum requirement) based on their access rights: (a) administrators, (b) curators, and (c) guests. Apart from the physical persons, the back-end infrastructure of the application, the repository, can be considered as a secondary actor of the system. It acts as an external system providing a set of services, dealing basically with the storage/update/retrieval of both content and metadata, to the application in order to accomplish several user goals.

Figure 8 presents a Use Case Diagram, as an overview of the EuroMuse functionality illustrating the actors of the system, their goals (represented as use cases), as well as any dependencies between those goals.

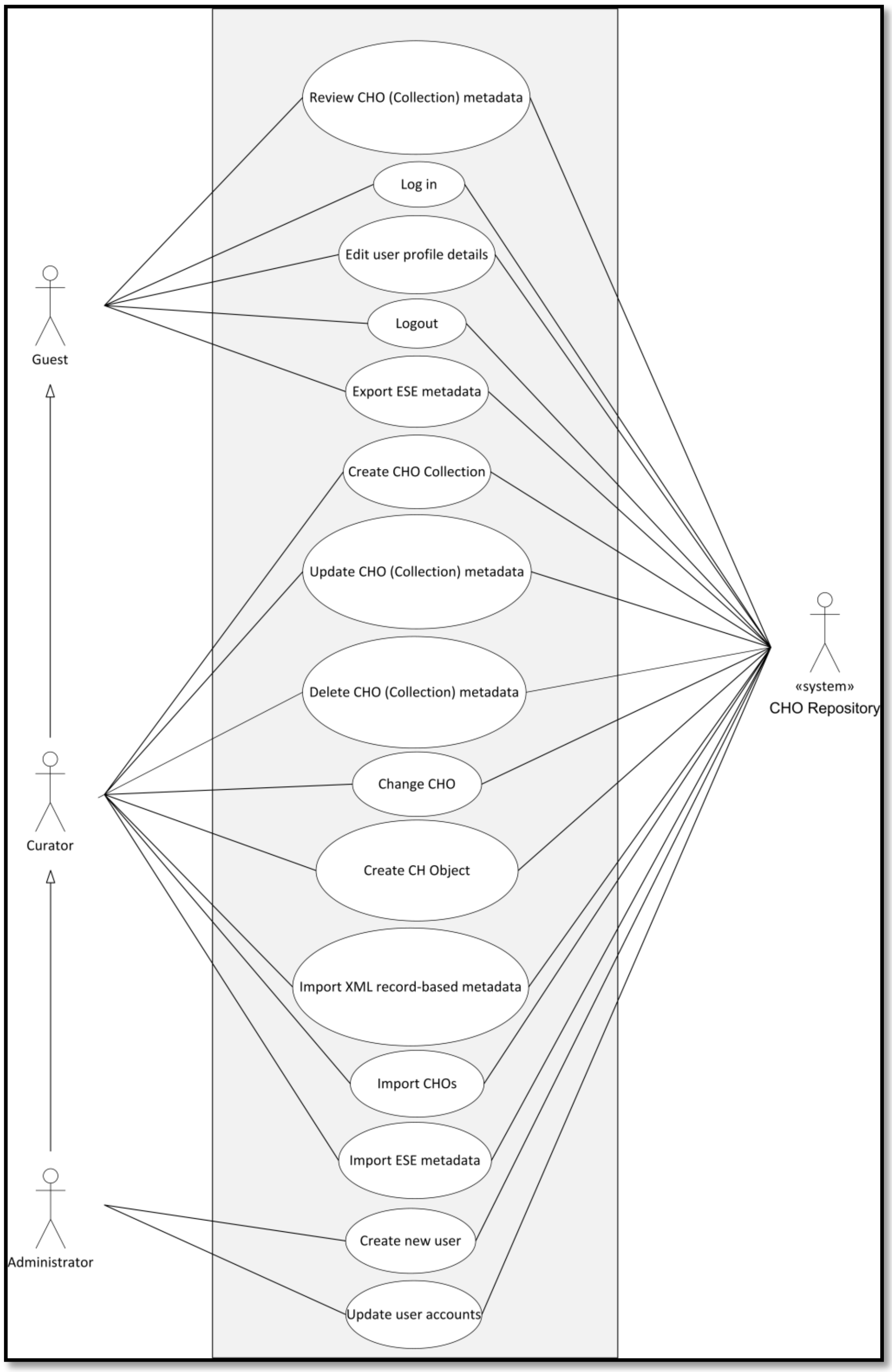


Figure 8: Use Case Diagram of the application.

Use Case 1	“Log in”	
Goal In Context	A user wants to be logged in the system in order to use the system’s services.	
Scope & Level	System, Sub-function	
Preconditions	The user has already an account to the system.	
Success End Condition	The system authenticates the user and the user logs in the system.	
Failed End Condition	The user couldn’t be logged in the system.	
Primary, Secondary Actors	Guest, Curator, Administrator	
Trigger	The user visits the application url without being already logged in.	
Description	**Step**	**Action**
	1	The system displays the log in form.
	2	User fills in the form with his credentials and selects to log in.
	3	The system validates the submitted form details, checks if the user credentials are correct.
	4	The system displays the first page according to the type of the user’s account.
Extensions	**Step**	**Branching Action**
	3a	There is another user already logged in the system with the same account. **1.** The system informs the user that there is another user already logged in. **2.** Use case fails.
	3b	The user credentials are incorrect. **1.** The system informs the user that the username or password is incorrect and prompts him to try again. **2.** Use case fails.

Use Case 2	“Logout”	
Goal In Context	The user wants to be logged out from the system.	
Scope & Level	System, Sub-function	
Preconditions	The user is already logged in the system.	
Success End Condition	The user is logged out from the system.	
Failed End Condition	The user couldn’t be logged out from the system.	
Primary, Secondary Actors	Guest, Curator, Administrator	
Trigger	The user selects to logout from the system.	
Description	**Step**	**Action**
	1	The system logs the user out of the system.
	2	The system displays the login screen.

Use Case 3	**“Create CHO Collection”**	
Goal In Context	The user wants to create a new CHO collection.	
Scope & Level	System, Sub-function	
Preconditions	The user has already logged in the system and his user class is curator or administrator.	
Success End Condition	The user creates a new collection.	
Failed End Condition	No new collection has been created.	
Primary, Secondary Actors	Curator, Administrator	
Trigger	The user selects to create a new collection using the application menu.	
Description	**Step**	**Action**
	1	The system asks the user for the new collection’s name.
	2	The user enters the name of the collection.
	3	The system validates the user’s input.
	4	The system creates the collection with the given name and adds it to the collection list.
	5	The system informs the user about the creation of the collection.
	6	The system shows the metadata of the collection (use case 9).
Extensions	**Step**	**Branching Action**
	2a	The collection name is empty. 10.1. The system informs the user about the error, and asks him to complete the name field. 10.2. Use case stays at step 2 until a valid collection name is inserted.

Use Case 4	**“Create CH Object”**	
Goal In Context	User wants to create an object in an existing collection.	
Scope & Level	System, Sub-function	
Preconditions	There is already at least one collection in the system, the user is already logged in and he is a curator or administrator.	
Success End Condition	The user creates a new object.	
Failed End Condition	The user couldn’t create a new object.	
Primary, Secondary Actors	Curator, Administrator	
Trigger	The user selects to create a new object using the application menu.	
Description	**Step**	**Action**
	1	The system checks if there is a selected collection.
	2	The system updated the collection list with a new object under the selected collection.
	3	The system displays the object’s metadata (all the elements are blank when the object is first created).
	4	The user fills in the metadata elements and saves the object.
	5	The system checks if all required fields have been completed by the user.
	6	The system creates a new object and notifies the user.
	7	The system displays the newly created object with its metadata.
Extensions	**Step**	**Branching Action**
	1a	User hasn’t selected any collection. **1.** The system informs the user.
	5a	The required fields were not completed. **1.** The system informs the user and displays the object’s metadata.

Use Case 5	**“Import record-based XML metadata”**	
Goal In Context	User wants to import legacy metadata into the system.	
Scope & Level	System, Sub-function	
Preconditions	The user is already logged in as Administrator/Curator, and the metadata are in record-based xml format.	
Success End Condition	The user imports the metadata in the system.	
Failed End Condition	The metadata have not been imported.	
Primary, Secondary Actors	Curator, Administrator	
Trigger	The user selects the Metadata Import option from the application menu.	
Description	**Step**	**Action**
	1	The system displays a number of options for the import process (e.g. files for upload).
	2	The user fills in the form with the requested information and submits them.
	3	The system transforms the imported metadata to ESE format using the mappings provided, and creates the new CHO metadata records.
	4	The system notifies the user about the result of the import.
Extensions	**Step**	**Branching Action**
	3a	An error occurred during the transformation of the metadata. **1.** The system informs the user. **2.** Use case fails.

Use Case 6	**"Import ESE metadata"**	
Goal In Context	User wants to import ESE metadata into the system.	
Scope & Level	System, Sub-function	
Preconditions	The user is already logged in as Administrator/Curator, and the metadata are in valid ESE format.	
Success End Condition	The user imports the metadata in the system.	
Failed End Condition	The metadata have not been imported.	
Primary, Secondary Actors	Curator, Administrator	
Trigger	The user selects the ESE Metadata Import option from the application menu.	
Description	**Step**	**Action**
	1	The system displays a number of options for the import process.
	2	The user fills in the form with the requested information and submits them.
	3	The system validates the imported metadata, and creates the new CHO metadata records.
	4	The system notifies the user about the result of the import.
Extensions	**Step**	**Branching Action**
	3a	An error occurred during the validation of the metadata. **1.** The system informs the user. **2.** Use case fails.

Use Case 7	**“Import CH Object”**	
Goal In Context	User wants to import CH objects into the system.	
Scope & Level	System, Sub-function	
Preconditions	The user is already logged in as Administrator/Curator.	
Success End Condition	The user imports the CH objects in the system.	
Failed End Condition	The CH objects have not been imported.	
Primary, Secondary Actors	Curator, Administrator	
Trigger	The user selects the “File->Import->Media Objects” option from the menu bar.	
Description	**Step**	**Action**
	1	The system displays a number of options for the import process.
	2	The user fills in the form with the requested information and submits them.
	3	The system validates the imported data, and creates the new CHOs.
	4	The system notifies the user about the result of the import.
Extensions	**Step**	**Branching Action**
	3a	An error occurred during the import of the objects. **1.** The system informs the user. **2.** Use case fails.

Use Case 8	**"Export ESE metadata"**	
Goal In Context	User wants to export the metadata of a collection/record in ESE format.	
Scope & Level	System, Sub-function	
Preconditions	The user is already logged in as Administrator/Curator.	
Success End Condition	The user has successfully exported the metadata.	
Failed End Condition	The metadata have not been exported.	
Primary, Secondary Actors	Guest, Curator, Administrator	
Trigger	The user selects the Export ESE Metadata option from the menu bar.	
Description	**Step**	**Action**
	1	The user selects the collection or record from the Collection Browser Widget, for which he wished the metadata to be exported.
	2	The system converts the metadata to ESE format and prompts the user to save the produced xml file.
	3	The user saves the metadata file to his computer.
Extensions	**Step**	**Branching Action**
	2a	An error occurred during the export of the metadata. **1.** The system informs the user. **2.** Use case fails.

Use Case 9	**<u>“Review CHO Collection metadata”</u>**	
Goal In Context	User wants to review the metadata of a collection.	
Scope & Level	System, Sub-function	
Preconditions	The user is already logged in and there are CHO Collections already in the system.	
Success End Condition	The user reviews the metadata of the CHO Collection.	
Failed End Condition	The user couldn’t review the CHO Collection’s metadata.	
Primary, Secondary Actors	Guest, Curator, Administrator	
Trigger	The user selects a CHO Collection from the collection list.	
Description	**Step**	**Action**
	1	The system checks if the previously reviewed CHO Collection/CHO has unsaved changes.
	2	The system displays the selected CHO Collection’s metadata.
Extensions	**Step**	**Branching Action**
	1a	The previously reviewed CHO Collection/CHO has unsaved changes. **1.** The system informs the user and prompts him to save or discard the changes before continuing.
	1a	The CHO Collection has been deleted by another user. **1.** The system informs the user.

Use Case 10	**“Review CHO metadata”**	
Goal In Context	User wants to review the metadata of a CHO.	
Scope & Level	System, Sub-function	
Preconditions	The user is already logged in and there are CHO metadata records already in the system.	
Success End Condition	The user reviews the metadata of the record.	
Failed End Condition	The user couldn’t review the record metadata.	
Primary, Secondary Actors	Guest, Curator, Administrator	
Trigger	The user selects a record from the collection list.	
Description	**Step**	**Action**
	1	The system checks if the previously reviewed collection/record has unsaved changes.
	2	The system displays the selected record’s metadata.
Extensions	**Step**	**Branching Action**
	1a	The previously reviewed collection/record has unsaved changes. **1.** The system informs the user and prompts him to save or discard the changes before continuing.
	1a	The record has been deleted by another user. **1.** The system informs the user.

Use Case 11	**“Update CHO Collection metadata”**	
Goal In Context	User wants to update the metadata of a collection.	
Scope & Level	System, Sub-function	
Preconditions	The user is already logged in and he is curator or administrator.	
Success End Condition	The user updates the metadata of the collection successfully.	
Failed End Condition	The user couldn’t update the metadata.	
Primary, Secondary Actors	Curator, Administrator	
Trigger	The user selects to Update an already open collection.	
Description	**Step**	**Action**
	1	The reviews a collection from the collection list (use case 9).
	2	The user selects to edit the open collection.
	3	The system displays all the metadata elements available, based on the application profile.
	4	The user edits/adds/deletes the metadata information and saves the changes.
	5	The system checks the user’s input and sees if all required fields have been completed.
	6	The system updates the collection and notifies the user.
	7	The system displays the object’s updated metadata.
Extensions	**Step**	**Branching Action**
	3a.	Another user is already editing the collection **1.** The system informs the user. **2.** The user has to wait until the editing of the collection has been completed by the other user.
	4a	The required fields were not completed. **1.** The system informs the user. **2.** User corrects the input before continuing the use case.

Use Case 12	**“Update CHO metadata”**	
Goal In Context	User wants to update the metadata of a CHO.	
Scope & Level	System, Sub-function	
Preconditions	The user is already logged in and he is curator or administrator.	
Success End Condition	The user updates the metadata of the CHO successfully.	
Failed End Condition	The user couldn’t update the metadata.	
Primary, Secondary Actors	Curator, Administrator	
Trigger	The user selects to update an already open record.	
Description	**Step**	**Action**
	1	The user reviews a CHO from the collection list (use case 10).
	2	The user selects to edit the open CHO.
	3	The system displays all the metadata elements available, based on the application profile.
	4	The user edits/adds/deletes the metadata information and saves the changes.
	5	The system checks the user’s input and sees if all required fields have been completed.
	6	The system updates the CHO and notifies the user.
	7	The system displays the object’s updated metadata.
Extensions	**Step**	**Branching Action**
	3a.	Another user is already editing the object. 1. The system informs the user. 2. The user has to wait until the editing of the object has been completed by the other user.
	4a	The required fields were not completed. **1.** The system informs the user. **2.** User corrects the input before continuing the use case.

Use Case 13	**<u>“Delete CHO Collection”</u>**	
Goal In Context	User wants to delete a CHO collection.	
Scope & Level	System, Sub-function	
Preconditions	The user is already logged in and he is curator or administrator.	
Success End Condition	The user deletes the collection successfully.	
Failed End Condition	The user couldn’t delete the collection.	
Primary, Secondary Actors	Curator, Administrator	
Trigger	The user selects to Delete an already open collection.	
Description	**Step**	**Action**
	1	The selects to update a collection from the collection list (use case 11).
	2	The user selects to delete the open collection.
	3	The system deletes the collection and notifies the user.

Use Case 14	**<u>“Delete CH Object”</u>**	
Goal In Context	User wants to delete a CHO.	
Scope & Level	System, Sub-function	
Preconditions	The user is already logged in and he is curator or administrator.	
Success End Condition	The user deletes a CHO successfully.	
Failed End Condition	The user couldn’t delete the CHO.	
Primary, Secondary Actors	Curator, Administrator	
Trigger	The user selects to delete an already open record.	
Description	**Step**	**Action**
	1	The user edits a CHO from the collection list (use case 12).
	2	The user selects to delete the open CHO.
	3	The system deletes the CHO and notifies the user.

Use Case 15	**“Change CHO”**	
Goal In Context	User wants to change the media object of the CHO.	
Scope & Level	System, Sub-function	
Preconditions	The user is already logged in as Administrator/Curator.	
Success End Condition	The media object of the CHO has been changed.	
Failed End Condition	The media object has not been changed.	
Primary, Secondary Actors	Curator, Administrator	
Trigger	The user selects the Change Media Object option from the menu.	
Description	**Step**	**Action**
	1	The user selects a CHO from the CHO Collection list for update (use case 12) and selects to change its media object from the menu.
	2	The system asks the user to choose from already published or unpublished media file and complete the corresponding form.
	3	The user inserts the URI of the object if it is already published on the web, or the media file from his pc and submits the form.
	4	The system creates thumbnails for the media object and publishes the file if it is unpublished.
	5	The system displays the newly created thumbnail.
	6	User saves the updated objet in order for the change to complete.
Extensions	**Step**	**Branching Action**
	4a	An error occurred during the creation of the thumbnail or the publishing of the object. **1.** The system informs the user. **2.** Use case fails.

Use Case 16	**"Create new user"**	
Goal In Context	The user creates a new user of the system.	
Scope & Level	System, User Level	
Preconditions	The user is already logged in and he has administrative privileges.	
Success End Condition	The user creates a new user successfully.	
Failed End Condition	No new user has been created.	
Primary, Secondary Actors	Administrator	
Trigger	The user selects to create a new user.	
Description	**Step**	**Action**
	1	The system displays a form with the fields required for the creation of a user.
	2	The user fills in the form and submits the new user details to the system.
	3	The system validates the user's input.
	4	The system creates the new user and displays a confirmation.
Extensions	**Step**	**Branching Action**
	3a	The required fields were not completed, or the input was invalid. • The system informs the user • The user has to correct the input before continuing.

Use Case 17	**“Update user accounts”**	
Goal In Context	The user wants to update the user accounts.	
Scope & Level	System, Sub-function	
Preconditions	The user is already logged in and he has administrative privileges.	
Success End Condition	The user updates the accounts successfully.	
Failed End Condition	The user couldn’t update the accounts.	
Primary, Secondary Actors	Administrator	
Trigger	The user selects to update the user accounts from the administrative menu.	
Description	**Step**	**Action**
	1	The system displays a list with all the users’ account details.
	2	The user updates any field from any account and selects to save the changes.
	3	The system validates the user’s input.
	4	The system saves the changes and updates the list of the user account details.
Extensions	**Step**	**Branching Action**
	3a	The required fields were not completed, or the input was invalid. • The system informs the user • The user has to correct the input before continuing.

Chapter 4

The ESE-CHO Application Profile

This chapter presents the ESE-CHO Application Profile that will serve as the basis for the description of the Cultural Heritage Objects (CHOs) that will be populated into the Natural Europe repositories using our application. The application profile has been based on the Europeana Semantic Elements specification.

The ESE-CHO Application Profile consists of the following parts:

- The *basic information*, that groups the general information that describes a CHO as a whole
- The *Life Cycle information*, that describes the history and current state of a CHO and those entities that have affected this object during its evolution using the EuroMuse application(as well as the status of this CHO within the underlying repository)
- The *Technical information*, that describes the technical requirements and characteristics of a CHO
- The *Relation information,* that defines the relationship between a CHO and other CHOs, if any
- The *Collection information*, that provides metadata information for logical groupings of contributed CHOs within a museum
- The *Europeana information*, that provides various Europeana related metadata information needed for describing this CHO.

In the next sections we provide the various elements that belong to the above-mentioned information parts based on their type (mandatory, recommended, optional) and not the information part where they belong to. In addition, some elements are characterised as mandatory although they are automatically generated (i.e. without user intervention). For each element we also provide a short table that contains its Namespace and Datatype as well as the method of its input (manual, automatic), and indicative examples of element value(s) for both the GUI of the application and its internal representation. Last, for each mandatory/recommended element we also indicate the usage (way of exploitation) of this element in the Europeana.eu portal.

Table 6 summarises the elements referring to CH Objects, grouped by importance, as appear in the Europeana/ESE specification and in our ESE-CHO Application Profile. It also provides the element mappings between the two specifications.

Europeana/ESE Mandatory elements	**ESE-CHO Mandatory elements for CHOs**
dc:title or dc:description	Title
	Description
dc:language for text objects	Language. Strongly Recommended data element where available. Mandatory data element if element Type is "Text"
dc:subject or dc:type or dc:coverage or dcterms:spatial	Subject
	Spatial Coverage
	Date (or refinements: Date Created, Date Issued)
europeana:isShownAt or europeana:isShownBy	Context URL or Object URL
	Thumbnail URL
europeana:type	Content Type
europeana:dataProvider	Data provider
	Country
europeana:provider	Provider

europeana:rights	Licence
	Version
	Accessibility
	Status

Europeana/ESE Recommended elements	**ESE-CHO Recommended elements for CHOs**
	Classification
dcterms:alternative	Alternative Title
dc:creator	Creator
dc:contributor	Contributor
dc:publisher	Publisher
dc:type	Type
europeana:object	See above (Thumbnail URL)
dc:date	see above (Date)
dcterms:created	see above (Date Created)
dcterms:issued	see above (Date Issued)
dcterms:temporal	Temporal Coverage
dc:coverage	Coverage
dc:source	Source
dcterms:isPartOf	Is Part Of

Values supplied by Europeana	
europeana:language	
europeana:uri	
europeana:year	

Table 6: Summary of ESE/ESE-CHO elements

Table 7 summarises the elements referring to CHO Collections, grouped by importance, as appear in our ESE-CHO Application Profile. These elements are not mapped to any ESE elements as the ESE specification does not define any metadata elements for the description of CHO Collections.

ESE-CHO Mandatory elements for CHO Collections
Title
Creator
Subject
ESE-CHO Recommended elements for CHO Collections
Description
Contributor
Type
Identifier
Coverage

Table 7: Summary of ESE-CHO Collection elements

4.1. Data types

Four data types are considered:

String: the text can be entered in the element directly (e.g. “ISBN”, “Natural Europe”, “canis lupus”, etc.).

LangString: the text must identify its language and there can be one or more character strings in the element (e.g. “en”, “natural history”).

DateTime: the element contains date and time information and there can also be textual information about this point in time (e.g. “2008-07-21”, “iron age”).

VocabularyTerm: the element contains a source and a value part, where source is a reference to sourced and maintained value set and value is a literal value from that set (e.g. “http://www.catalogueoflife.org/details/species/id/6850415 ”, “Canis lupus Linnaeus, 1758”).

4.2. Mandatory elements for a CH Object

We consider as mandatory elements those elements that a user is obliged to fill in order to complete the description of the considered CH Object. The ESE considers that in order to accommodate different object types and different metadata practices, some mandatory elements are grouped together and only one in the group needs to be provided; but ideally all elements in a group would be provided. This application profile described here adopts this suggestion and adopts all elements in such a group as mandatory ones.

4.2.1. Title

The title or name by which the digital object is known. This is likely to be the title or name also applied to the original physical object in the case of a digitisation. Use the refinement *Alternative Title* for any title variants, translations etc. In case where the value in element *Classification* is obtained as a term from the proposed classification scheme then the value of the *Title* element is automatically filled with the preferred label of this term. User always has the option to override this title and/or manually insert his/her own terms.

	title
Namespace	dc
Datatype	LangString
Method of input	Automatic if chosen from element *Classification*, then manually (User always has the option to

	override)
Example (gui)	(“lat”, “Mammuthus primigenius”) (“eng”, “Wolf drinking water in natural habitat”)
Schema	“Mammuthus primigenius”@lat “Wolf drinking water in natural habitat”@eng
Use/Exploitation in the Europeana.eu portal	Simple search, Advanced search, Full search result display (Title Line)

Table 8: ESE-CHO Title element overview.

4.2.2. Description

A prose description of the digital object or the original physical object in the case of a digitisation, elaborating on the information in the metadata. The following are examples of data that could be mapped to the description element:

- statements relating to a technique applied to an object in terms of technology
- statements where a technique includes reference to a material.
- statements about an event relating to an object

	description
Namespace	dc
Datatype	LangString
Method of input	Manual
Example (gui)	(“en”, “The molar tooth of woolly mammoth, dated back to about 10 000-10 500 radiocarbon years is one of youngest mammoth finds in Europe.”) (“et”, “Puurmani lähistelt leitud karvase mammuti purihammas on Euroopa üheks noorimaks mammutileiuks. Selle vanuseks on määratud 10 000-10 500 radiosüsiniku aastat.”)

Schema	"The molar tooth of woolly mammoth, dated back to about 10 000-10 500 radiocarbon years is one of youngest mammoth finds in Europe."@en
Use/Exploitation in the Europeana.eu portal	Simple search, Full search result display (Description Line)

Table 9: ESE-CHO Description element overview.

4.2.3. Language

A value in this element is mandatory for "TEXT" objects and strongly recommended for other objects that have a language component, such as image legends. This element should be used to state the language of the digital object and should be repeated if the object has more than one language. If there is no language aspect to the object (for instance, a photograph) then the element should be ignored. The value(s) of this element should conform to ISO 639-2 (the three character code) specification. This element is not used to indicate the language of the metadata.

	language
Namespace	dc
Datatype	String
Method of input	Manual, based on the provided ISO 639-2 specification
Example (gui)	("est")
Schema	"est"
Use/Exploitation in the Europeana.eu portal	Simple search, Facet (Language), Full search result display (Language Line)

Table 10: ESE-CHO Language element overview.

4.2.4. Subject

The subject of the digital object or the original physical object in the case of a digitisation. This can include topics, people and places but consider using the spatial and temporal elements for places and time periods if the source data allows. Best practice is to use a separate instance of the element for individual subject terms. In case where the value in element *Classification* is obtained as a concept from the proposed classification scheme then the value of the subject element is automatically filled with the classification hierarchy preferred label(s) of this term. In this case the value(s) of this element are also terms of the NE SKOSified classification scheme. User always has the option to override these terms and/or manually insert his/her own terms.

	1.8 subject
Namespace	naturaleurope
Datatype	VocabularyTerm
Method of input	Automatic if chosen from element 1.11 classification, then manually (User always has the option to override)
Example (gui)	(“animalia”) (“chordata”) (“mammalia”) (“proboscidea”) (“elephantidae”)
Schema	“animalia”, “col:2362377” “chordata”, “col:2362754” “mammalia”, “col:2362755” “proboscidea”, “col:2362869” “elephantidae”, “col:2362870”
Use/Exploitation in the Europeana.eu portal	Simple search, Advanced search, Full search result display (Subject Line)

Table 11: ESE-CHO Subject element overview.

4.2.5. Version

The edition of a CH object during its evolution while it is managed in the context of Natural Europe using EuroMuse.

	2.1 version
Namespace	naturaleurope
Datatype	String
Method of input	Manual
Example (gui)	(“1.2.alpha”)
Schema	“1.2.alpha”
Use/Exploitation in the Europeana.eu portal	Not for use by Europeana.eu portal

4.2.6. Status

The completion status or condition of a CH object during its evolution while it is managed using our application. The value(s) in this element are terms chosen from a controlled vocabulary that is provided.

	status
Namespace	naturaleurope
Datatype	VocabularyTerm
Method of input	Manual, based on the provided vocabulary
Example (gui)	(“Editing in progress”)
Schema	“EDITING_IN_PROGRESS”
Use/Exploitation in the Europeana.eu portal	Not for use by Europeana.eu portal

Table 12: ESE-CHO Status element overview.

The vocabulary terms provided are the following:

- EDITING_IN_PROGRESS
- EDITING_COMPLETE
- REVIEW_IN_PROGRESS
- COMPLETE

4.2.7. Object URL

This element will be active in the Europeana.eu portal and will provide a link to the digital object on the provider website. This is a complementary element to element *Context URL* (below) and it is mandatory to provide a URL link in one of these elements. To map to this element the object should be directly accessible by the URL and reasonably independent at that location. The inclusion of, for example, short copyright information or minimal navigation tools would be acceptable in this element. The link should be to the object in its best available resolution/quality. Our application provides the necessary functionality in order to publish the object if not already published.

	uri
Namespace	naturaleurope
Datatype	URI
Method of input	Automatic whenever possible using the functionality of the application. For harvested material the information is automatically imported.
Example (gui)	(“http://geokogud.info/elm/specimen_image/g319/g319-2_a.jpg”)
Schema	“http://geokogud.info/elm/specimen_image/g319/g319-2_a.jpg”
Use/Exploitation in the Europeana.eu portal	Full search result display

Table 13: ESE-CHO Object URL element overview.

4.2.8. Context URL

This element will be active in the Europeana.eu portal and will provide the link to the digital object in full information context on the provider website. It is a complementary element to element *Object URL* and it is mandatory to provide a URL link in one of these elements. If the digital object is displayed with local metadata, with a header or banner, or if the object is only accessible by clicking another icon on the local page then the Context URL element should be used. Our application provides the necessary functionality in order to publish the object if not already published.

	contextUri
Namespace	naturaleurope
Datatype	URI
Method of input	Automatic whenever possible using the functionality of the application. For harvested material the information is automatically imported.
Example (gui)	(“http://geokogud.info/elm/specimen_image.php?id=690”)
Schema	“http://geokogud.info/elm/specimen_image.php?id=690”
Use/Exploitation in the Europeana.eu portal	Full search result display

Table 14: ESE-CHO Context URL element overview.

4.2.9. Thumbnail URL

This element supports the process of creating small images (thumbnails) for use in the Europeana.eu portal. This element should provide a link to the URL of a thumbnail representing the digital object or, if there is no such thumbnail, the URL of the digital object in the best resolution available on the web site of the data provider from which a thumbnail could be generated. Our application provides the necessary functionality in order to automatically create the thumbnail of a published object.

	sourceThumbnailUri
Namespace	naturaleurope

Datatype	URI
Method of input	Automatic whenever possible using the functionality of our application. For harvested material the information is automatically imported.
Example (gui)	(“http://147.27.41.103:8080/exist/rest//db/NHMRepository/content/thumbs/src/1klc1jqvrc9bmhf7glun3oa7b9”)
Schema	“http://147.27.41.103:8080/exist/rest//db/NHMRepository/content/thumbs/src/1klc1jqvrc9bmhf7glun3oa7b9”
Use/Exploitation in the Europeana.eu portal	Thumbnails

Table 15: ESE-CHO Thumbnail URL element overview.

4.2.10. Content Type

This element is used to support the Type facet in the Europeana.eu portal, the categorisation of objects in the result display and assignment of the appropriate default icon (if necessary). Europeana currently handles only four object types and uses a controlled vocabulary of four words expressed in upper case as the value of this element: TEXT, IMAGE, SOUND, and VIDEO.

	resourceType
Namespace	naturaleurope
Datatype	VocabularyTerm
Method of input	Automatic whenever possible. User always has the option to override. For harvested material the information is simply imported or if not given tried to be automatically detected.
Example (gui)	(“IMAGE”)
Schema	“IMAGE”

Use/Exploitation in the Europeana.eu portal	Simple search, Facet (Type), Advanced search, Full search result display (Type Line)

Table 16: ESE-CHO Content Type element overview.

4.2.11. Date

This date element should be used to contain the most significant date in the life of the digital object or the original physical object in the case of a digitisation. We recommend the use of ISO 8601 starting with the year and hyphenating the day and month parts: YYYY-MM-DD. As there are many ways of expressing dates and time periods and the values will display in the full record in whatever form they are supplied. The values will also be used as the basis for locating the object in the Timeline and the Date facet in the Europeana.eu portal and this must be a machine-readable date. If you are using the more precise date terms of dcterms:created or dcterms:issued these will also be used for the Timeline and Date facet. Currently, the Europeana portal cannot use Before Christ, Before Common Era or Before Present dates but such dates should be retained in the mapped metadata (i.e. dc:date) in order to be present for future development of the Europeana.eu portal. Textual time periods will display in a result list but cannot be represented in the Timeline or Date facet and should also be provided as numeric dates.

	date
Namespace	dc
Datatype	LangString
Method of input	Manual
Example (gui)	(“2008-05-16”) (“1601”, “1700”) (“17th century”)
Schema	“2008-05-16” “1601”, “1700” “17th century”
Use/Exploitation in the Europeana.eu portal	Simple search, Advanced search, Full search result display (Date Line)

Table 17: ESE-CHO Date element overview.

4.2.12. Date Created

This is the date of the creation of the digital object or, in the case of a digitisation, the original physical object. A refinement of the element *Date* (see above for fuller information about the form of the date).

	created
Namespace	dcterms
Datatype	LangString
Method of input	Manual
Example (gui)	(“2008-05-16”) (“1601”, “1700”) (“Iron Age”)
Schema	“2008-05-16” “1601”, “1700” “Iron Age”
Use/Exploitation in the Europeana.eu portal	Simple search, Full search result display (Date Line)

Table 18: ESE-CHO Date Created element overview.

4.2.13. Date Issued

The date when the digital object was formally issued or published. This is likely to be the date the original physical object was issued in the case of a digitisation. A refinement of the element *Date* (see above for fuller information about the form of the date).

	issued
Namespace	dcterms
Datatype	LangString
Method of input	Manual

Example (gui)	("2008-05-16") ("1601")
Schema	"2008-05-16" "1601"
Use/Exploitation in the Europeana.eu portal	Simple search, Full search result display (Date Line)

Table 19: ESE-CHO Date Issued element overview.

4.2.14. Spatial Coverage

This element is a refinement of element *Coverage*. Information about the spatial characteristics of the original analogue or born digital object, i.e. what the resource represents or depicts in terms of space. This may be a named place, a location, a spatial coordinate or a named administrative entity. It is preferred to use this more specific element for information about the spatial characteristics of the digital object or the original physical object in the case of a digitisation. Values in this element will appear in the Geographic coverage line of a full result display in the Europeana.eu portal.

	spatial
Namespace	dcterms
Datatype	LangString
Method of input	Manual
Example (gui)	("en", "Puurmani") ("en", "Estonia") ("et", "Puurmani") ("et", "Eesti")
Schema	"Puurmani"@en "Estonia"@en "Puurmani"@et "Eesti"@et

Use/Exploitation in the Europeana.eu portal	Simple search, Advanced search, Full search result display (Geographic coverage Line)

Table 20: ESE-CHO Spatial Coverage element overview.

4.2.15. Data Provider

Most of the data sent to Europeana comes via Aggregators whose names are recorded in the *Provider* element (element *Provider* supports one of the facets in the portal). This element, *DataProvider*, is specifically introduced to allow the names of the organisations who supply the data to an aggregator to be unambiguously recorded. The name should be the preferred form of the name in the language the provider chooses as the default language for display in the Europeana.eu portal. There can only be one instance of this element so countries desiring to show the name in multiple languages may concatenate the different language versions, for example: *"Estonian Museum of Natural History / Eesti Loodusmuuseum"*

The name that is provided for element *DataProvider* is not necessarily the institution where the physical object is located. This element is automatically filled during the ingestion process using the official name (see above) of the provider of a dataset.

	dataProvider
Namespace	europeana
Datatype	String
Method of input	Automatic, filled during the ingestion phase
Schema	"Estonian Museum of Natural History/ Eesti Loodusmuuseum"
Use/Exploitation in the Europeana.eu portal	Simple search, Advanced search, Full search result display (Date Provider Line)

Table 21: ESE-CHO Data Provider element overview.

4.2.16. Country

This element is used to support the Country facet in the Europeana.eu portal. The country is associated with the provider of a dataset.

	country
Namespace	europeana
Datatype	String
Method of input	Automatic, filled during the ingestion phase
schema	"ESTONIA"
Use/Exploitation in the Europeana.eu portal	Simple search, Facet, Full search result display (Country Line)

Table 22: ESE-CHO Country element overview.

4.2.17. Provider

This element supports the Provider facet in the Europeana.eu portal and should contain the name of the organisation that delivers data directly to Europeana. In Natural Europe project this will be the name of an aggregator or the project's name.

	provider
Namespace	europeana
Datatype	String
Method of input	Automatic, filled during the ingestion phase
Schema	"The Natural Europe Project"
Use/Exploitation in the Europeana.eu portal	Simple search, Facet, Full search result display (Provider Line)

Table 23: ESE-CHO Provider element overview.

4.2.18. Licence

The value in this element will indicate the usage and access rights that apply to the digital object described in the metadata and to the small portal images (thumbnails) used in Europeana. The value is a URL in a controlled form and is used in the Europeana.eu portal to generate an appropriate badge for display beneath the preview of the object and as a search refinement. The URLs should be constructed according to the specifications in the "Rights

Guidelines". They are constructed by adding a code indicating the copyright status of an object to the domain name where that status is defined. The domains specified are europeana.eu and creativecommons.org. The value of this element is taken from a controlled vocabulary that is provided.

	8.4 licenceUri
Namespace	naturaleurope
Datatype	URI
Method of input	Manual based on the provided vocabulary
Example (gui)	("CC BY-NC")
Schema	"http://creativecommons.org/licenses/by-nc/3.0/"
Use/Exploitation in the Europeana.eu portal	Full search result display

The vocabulary terms provided are the following:

Europeana.eu domain

Rights Reserved - Free Access	http://www.europeana.eu/rights/rr-f/
Rights Reserved - Paid Access	http://www.europeana.eu/rights/rr-p/
Rights Reserved - Restricted Access	http://www.europeana.eu/rights/rr-r/
Unknown	http://www.europeana.eu/rights/unknown/

CreativeCommons domain

CC – Zero (universal)	http://creativecommons.org/publicdomain/zero/1.0/
CC BY (v3.0 Unported)	http://creativecommons.org/licenses/by/3.0/
CC BY-SA	http://creativecommons.org/licenses/by-sa/3.0/
CC BY-NC	http://creativecommons.org/licenses/by-nc/3.0/
CC BY-NC-SA	http://creativecommons.org/licenses/by-nc-sa/2.0/

CC BY-ND	http://creativecommons.org/licenses/by-nd/2.0/
CC BY-NC-ND	http://creativecommons.org/licenses/by-nc-nd/1.0/

Table 24: ESE-CHO Licence element overview.

4.2.19. Accessibility

The access level of a CH object during its evolution while it is managed using our application. Defines whether the CHO is accessible by the harvester using the OAI-PMH protocol. The value(s) in this element are terms chosen from a controlled vocabulary that is provided.

	accessibility
Namespace	naturaleurope
Datatype	VocabularyTerm
Method of input	Manual, based on the provided vocabulary
Example (gui)	(“private”)
Schema	“PRIVATE”
Use/Exploitation in the Europeana.eu portal	Not for use by Europeana.eu portal

Table 25: ESE-CHO Accessibility element overview.

The vocabulary terms provided are the following:

- PRIVATE
- PUBLIC

4.3. Recommended elements for a CH Object

We consider as recommended elements those elements that a user is strongly recommended to fill in order to provide an acceptable record annotation.

4.3.1. Classification

The accepted scientific name of species, minerals, fossils and prehistoric artefacts. When possible this Latin term should be preferably taken by the provided classification scheme. For the purposes of the project the classification scheme has been SKOSified, therefore each term is represented as a SKOS concept. The recorded value of this element along with its related concepts within the SKOSified classification scheme are used for the automatic filling of other elements; the element *Title* is filled with the Latin term of the species (preferred label) and the element *Subject* is filled with the classification hierarchy terms of the species (e.g. phylum, class, order, etc.). In any case the user can manually override the automatically filled information. The schema definition of this element includes the SKOS term preferred label and the SKOS concept uri.

	classification
Namespace	naturaleurope
Datatype	VocabularyTerm
Method of input	Manual (an auto complete process based on the proposed classification scheme is offered)
Example (gui)	("Cyprinus carpio")
Schema	"Cyprinus carpio", "col:8617805"
Use/Exploitation in the Europeana.eu portal	Not for use by Europeana.eu portal

Table 26: ESE-CHO Classification element overview.

4.3.2. Alternative Title

This can be any alternative title or name by which the digital object is known and will often be the name also applied to the original physical object in the case of a digitisation. It can include abbreviations or translations of the title, including common names (for the natural history domain).

	alternative
Namespace	dcterms

Datatype	LangString
Method of input	Manual
Example (gui)	(“en”, “woolly mammoth”) (“et”, “karvane mammut”)
Schema	“karvane mammut”@et
Use/Exploitation in the Europeana.eu portal	Simple search, Advanced search, Full search result display (Title Line)

Table 27: ESE-CHO Alternative Title element overview.

4.3.3. Creator

The name of the creator or creators of the original physical object or the born digital object. Names can include those of people, organisations or services. Map each name to a separate, repeated creator element if possible. Ideally choose a preferred form of name from an authority source. If you do not use an authority source, use a consistent form of the name e.g. Shakespeare, William. This element should not be confused with the element *Contributor*. In the case of an animal photograph one could consider as Creator the photographer and as Contributor the person that has (possibly) processed this photograph. In the case of an exhibit photograph one could consider as Creator the person that set up this exhibit and as Contributor the photographer of this exhibit. This is an indicative example. However and with respect to the Europeana.eu portal it seems that for most of the cases in the natural history domain (i.e. items already uploaded) there are not so many creator(s) but mainly only contributor(s).

	creator
Namespace	dc
Datatype	LangString
Method of input	Manual
Example (gui)	(“en”, “Shakespeare, William”)
Schema	“Shakespeare, William”@ en
Use/Exploitation in the Europeana.eu portal	Simple search, Advanced search, Full search result display (Creator Line)

Table 28: ESE-CHO Creator element overview.

4.3.4. Contributor

The name of contributors to either the original physical object or the born digital object. Names can include those of people, organisations or services. Map each name to a separate repeated contributor element if possible. Ideally choose a preferred form of name from an authority source. If you do not use an authority source, use a consistent form of the name e.g. Shakespeare, William. For each contributor the kind of contribution has to be provided as a role for the contributor. A number of roles are provided as vocabulary terms in the MARC Code List for Relators (http://id.loc.gov/vocabulary/relators.html).

	contributor
Namespace	naturaleurope
Datatype	LangString
Method of input	Manual
Example (gui)	(“Friedrich Schmidt (collector)”) (“Tiit Hunt (photographer)”)
Schema	“Tiit Hunt”, “photographer”, “roles:http://id.loc.gov/vocabulary/relators/pht”
Use/Exploitation in the Europeana.eu portal	Simple search, Advanced search, Full search result display (Creator Line)

Table 29: ESE-CHO Contributor element overview.

4.3.5. Publisher

The name of the publisher (the entity responsible for making the resource available) of the digital object or the original physical object in the case of a digitisation.

	publisher
Namespace	dc
Datatype	LangString
Method of input	Manual

Example (gui)	(“en”, “Estonian Museum of Natural History”) (“et”, “Eesti Loodusmuuseum”)
Schema	“Estonian Museum of Natural History”@en “Eesti Loodusmuuseum”@et
Use/Exploitation in the Europeana.eu portal	Simple search, Full search result display (Publisher Line)

Table 30: ESE-CHO Publisher element overview.

4.3.6. Type

The nature or genre of the digital object or the original physical object in the case of a digitisation. This should be used to record the values given in the source data which, ideally, will have been taken from a controlled vocabulary. Typically this element contains values such as photograph, painting and sculpture. A controlled DC Type vocabulary is provided (available at http://dublincore.org/documents/dcmi-type-vocabulary/).

	type
Namespace	naturaleurope
Datatype	LangString or VocabularyTerm
Method of input	Manual, preferably the values should be taken from the provided vocabulary
Example (gui)	(“en”, “subfossil”) (“et”, “subfossiil”) or (“Image”)
Schema	“subfossil” @en “subfossiil”@et or “Image”, “dcmitype:http://purl.org/dc/dcmitype/Image”

Use/Exploitation in the Europeana.eu portal	Simple search, Full search result display (Type Line)

Table 31: ESE-CHO Type element overview.

4.3.7. Coverage

Coverage can be used for either spatial or temporal aspects of the object being described. Values will typically include either a spatial location (place name or geographic co-ordinates) or a temporal period (a date range or period label). If analysis of the data shows that it contains only spatial or only temporal data then please map to either the *Spatial Coverage* or the *Temporal Coverage* element refinements described next: the greater precision will allow the addition of place and time-based functionality. If values in the source data are mixed or unknown then this more generic *Coverage* element should be chosen for the mapping.

	coverage
Namespace	dc
Datatype	LangString
Method of input	Manual
Example (gui)	(“en”, “Boston, MA”) (“en”, “1995-1996”)
Schema	“Boston, MA”@en “1995-1996”@en
Use/Exploitation in the Europeana.eu portal	Simple search, Full search result display (Coverage Line)

Table 32: ESE-CHO Coverage element overview.

4.3.8. Temporal Coverage

Use this element for the temporal characteristics of the digital object or the original physical object in the case of a digitisation i.e. what the resource is about or depicts in terms of time.

This is in contrast to element *Date* which relates to an event in the life of the object itself (e.g. the creation or the art work or publication of the book.). Values in this element will appear in the Time period line of a full result display in the Europeana.eu portal.

	temporal
Namespace	dcterms
Datatype	LangString
Method of input	Manual
Example (gui)	(“en”, “Pleistocene – Holocene”)
Schema	“Pleistocene – Holocene”@en
Use/Exploitation in the Europeana.eu portal	Simple search, Advanced search, Full search result display (Time period Line)

Table 33: ESE-CHO Temporal Coverage element overview.

4.3.9. Is Part Of

This element should be used to identify a related resource in which the described resource is physically or logically included. More particularly, use this for the name of the collection of which the digital object is part.

	isPartOf
Namespace	dcterms
Datatype	LangString
Method of input	Manual
Example (gui)	(“en”, “Collection of fish images by Tiit Hunt”) (“et”, “Tiit Hunt'i kalafotod”)
Schema	“Collection of fish images by Tiit Hunt”@en “Tiit Hunt'i kalafotod”@et
Use/Exploitation in the Europeana.eu	Simple search, Full search result display (Relation Line)

portal	

Table 34: ESE-CHO Is Part Of element overview.

4.3.10. Source

This element can be used for several different types of sources that are related to the object (such as reference sources). The name of the content holder should not be recorded here as a new element.

	source
Namespace	dc
Datatype	LangString
Method of input	manual
Example (gui)	(“en”, “BAM portal”) (“en”, “Security Magazine pp 3-12”)
Schema	“BAM portal”@en “Security Magazine pp 3-12”@en
Use/Exploitation in the Europeana.eu portal	Simple search, Full search result display (Source)

Table 35: ESE-CHO Source element overview.

4.4. Optional elements for a CH Object

4.4.1. Provenance

This element is to record a statement of any changes in ownership and custody of the resource since its creation that are significant for its authenticity, integrity and interpretation. This may include a description of any changes successive custodians made to the resource. For Europeana this relates particularly to the ownership and custody of the original analog or born-digital object.

	provenance
Namespace	dcterms
Datatype	LangString
Method of input	Manual
Example (gui)	(“en”, “Donated by the National Library in 1965”)
schema	“Donated by the National Library in 1965”@en

Table 36: ESE-CHO Provenance element overview.

4.4.2. Format

This element can include the file format, physical medium and dimensions of the original physical object or the digital object. It is recommended to use this element for the file format of digitised or born-digital object. For digital objects it is recommended to use the provided Internet media types (originally called MIME types) based on IANA registration. The refinements of *Extent* and *Medium* can be used as appropriate for the more specific information. In case that a user publishes an object using our application then (whenever possible) this element is automatically generated during the publishing process.

	format
Namespace	dc
Datatype	String
Method of input	Automatic whenever possible. User always has the option to override. For harvested material the information is simply imported or if not given tried to be automatically detected.
Example (gui)	(“image/jpeg”)
schema	“image/jpeg”

Table 37: ESE-CHO Format element overview.

4.4.3. Extent

Use to record the size or duration of the original physical or digital object. This includes measurements of physical objects that have been digitised. To ensure a meaningful display for the user, please indicate the units of measurement in the value. In case that a user publishes an object using our application then (whenever possible) this element is automatically generated during the publishing process.

	extent
Namespace	dcterms
Datatype	LangString
Method of input	Automatic whenever possible. User always has the option to override. For harvested material the information is simply imported or if not given tried to be automatically detected.
Example (gui)	('en", "1280 x 827 pixels") ("en", "42.4 cm x 68 cm")
Schema	"1280 x 827 pixels"@en "42.4 cm x 68 cm"@en

Table 38: ESE-CHO Extent element overview.

4.4.4. Medium

The material or physical carrier of the resource. This refers to the medium of the digital or original physical object such as paper, wood or ivory.

	medium
Namespace	dcterms
Datatype	LangString
Method of input	Manual
Example (gui)	("en", "metal")
Schema	"metal"@en

Table 39: ESE-CHO Medium element overview.

4.4.5. Identifier

This element can be used for an identifier of the digital object or the original physical object in the case of a digitisation.

	identifier
Namespace	dc
Datatype	String
Method of input	Manual
Example (gui)	(“http://geokogud.info/elm/specimen_image/g297/g297-13_b.jpg”) (“urn:isbn:9780387097466”)
Schema	“http://geokogud.info/elm/specimen_image/g297/g297-13_b.jpg” “urn:isbn:9780387097466”

Table 40: ESE-CHO Identifier element overview.

4.4.6. Rights

This element contains information about intellectual Property Rights, access rights or licence arrangements for the digital object (digitized or born digital). It’s value can be any additional information about intellectual property rights, access rights or license arrangements for the digital object that has not been captured in *Licence* element. In any case the elements *Rights* and *Licence* should not contain the same value.

	rights
Namespace	dc
Datatype	LangString
Method of input	Manual
Example (gui)	(“en”, “Estonian Museum of Natural History”) (“et”, “Eesti Loodusmuuseum”)

Schema	"Estonian Museum of Natural History"@en "Eesti Loodusmuuseum"@et

Table 41: ESE-CHO Rights element overview.

4.4.7. Table Of Contents

Use for a list of the sub-units of the digital object or the original physical object in the case of a digitisation.

	tableOfContents
Namespace	dcterms
Datatype	String
Method of input	Manual
Example (gui)	("Chapter 1. Introduction, Chapter 2. History")
Schema	"Chapter 1. Introduction, Chapter 2. History"

Table 42: ESE-CHO Table Of Contents element overview.

4.4.8. Relation

This element should be used for information about resources that are related to the digital object or the original physical object in the case of a digitisation. It has been used for a wide range of relationships and it is recommended to use one of the several more specific relationship refinements (follow below) where appropriate. Ideally this value should be a URI but it is recognised that practice varies in this respect.

	relation
Namespace	dc
Datatype	String
Method of input	Manual

Example (gui)	(“maps.crace.1/33”) *note: This is the shelf mark for a map held in the British Library's Grace Collection*
Schema	“maps.crace.1/33”

Table 43: ESE-CHO Relation element overview.

4.4.9. Conforms To

The names of standards that the digital object (digitized or born digital) complies with and which are useful for the use of the object.

	conformsTo
Namespace	dcterms
Datatype	String
Method of input	Manual
Example (gui)	(“W3C WCAG 2.0”) *note: For an HTML document that conforms to web content accessibility guidelines*
Schema	“W3C WCAG 2.0”

Table 44: ESE-CHO Conforms To element overview.

4.4.10. Has Format

Use this element to identify another resource that is substantially the same as the digital object being described by the metadata but exists in a different format. Note that the purpose of this element is to give the identifier of the other resource in a different format, not to state the format of the object being described.

	hasFormat
Namespace	dcterms
Datatype	String

Method of input	Manual
Example (gui)	("http://upload.wikimedia.org/wikipedia/en/f/f3/Europeana_logo.png") *note: A link to another image format of the tiff image file being described*
Schema	"http://upload.wikimedia.org/wikipedia/en/f/f3/Europeana_logo.png"

Table 45: ESE-CHO Has Format element overview.

4.4.11. Is Format Of

Opposite of *Has Format*. Use this element to identify a related resource that is substantially the same as the digital object but in a different format. Use when there are alternative formats and it is not clear which preceded the other.

	isFormatOf
Namespace	dcterms
Datatype	String
Method of input	Manual
Example (gui)	("Europeana_logo.tiff") note: Where the resource being described is a png image file
Schema	"Europeana_logo.tiff"

Table 46: ESE-CHO Is Format Of element overview.

4.4.12. Has Version

A related object that is a version, edition, or adaptation of the described object. Changes in version imply substantive changes in content rather than differences in format.

	hasVersion
Namespace	dcterms
Datatype	LangString
Method of input	Manual
Example (gui)	(“en”, “The Sorcerer's Apprentice”) *note: For the translation by Edwin Zeydel of Goethe's poem Der Zauberlehrling, where the metadata record is describing the original*
Schema	“The Sorcerer's Apprentice”@en

Table 47: ESE-CHO Has Version element overview.

4.4.13. Is Version Of

(The opposite of *Has Version*), a related object of which the described object is a version, edition, or adaptation. Changes in version imply substantive changes in content rather than differences in format.

	isVersionOf
Namespace	dcterms
Datatype	langString
Method of input	Manual
Example (gui)	(“en”, “ESE Version 0.5”)
Schema	“ESE Version 0.5”@en

Table 48: ESE-CHO Is Version Of element overview.

4.4.14. Has Part

A related object that is included either physically or logically in the described object.

	hasPart
Namespace	dcterms
Datatype	String
Method of input	Manual
Example (gui)	(“Maps.added.22231”) *note: The identifier for another map which is part of this one that is being described*
Schema	“Maps.added.22231”

Table 49: ESE-CHO Has Part element overview.

4.4.15. Is Referenced By

A related object that references, cites, or otherwise points to the described object.

	isReferencedBy
Namespace	dcterms
Datatype	LangString
Method of input	Manual
Example (gui)	(“en”, “Till, Nicholas (1994) Mozart and the Enlightenment: Truth, Virtue and Beauty in Mozart’s Operas, W. W. Norton & Company “)
Schema	“Till, Nicholas (1994) Mozart and the Enlightenment: Truth, Virtue and Beauty in Mozart’s Operas, W. W. Norton & Company “@en

Table 50: ESE-CHO Is Referenced By element overview.

4.4.16. References

(The opposite of *Is Referenced By*), a related object that is referenced, cited, or otherwise pointed to by the described object.

	references
Namespace	dcterms
Datatype	LangString
Method of input	Manual
Example (gui)	(“en”, “Security Magazine pp 3-12“)
Schema	“Security Magazine pp 3-12“@en

Table 51: ESE-CHO References element overview.

4.4.17. Is Replaced By

A related object that supplants, displaces, or supersedes the described object.

	isReplacedBy
Namespace	dcterms
Datatype	String
Method of input	Manual
Example (gui)	(“http://dublincore.org/about/2009/01/05/bylaws/”) *note: Where the resource described is an older version, say http://dublincore.org/about/2006/01/01/bylaws/*
Schema	“http://dublincore.org/about/2009/01/05/bylaws/”

Table 52: ESE-CHO Is Replaced By element overview.

4.4.18. Replaces

(The opposite of *Is Replaced By*), a related object that is supplanted, displaced, or superseded by the described object.

	replaces
Namespace	dcterms
Datatype	LangString
Method of input	Manual
Example (gui)	("http://dublincore.org/about/2006/01/01/bylaws/") *note: Where the resource described is a newer version, say http://dublincore.org/about/2009/01/05/bylaws/*
Schema	"http://dublincore.org/about/2006/01/01/bylaws/"

Table 53: ESE-CHO Replaces element overview.

4.4.19. Is Required By

A related object that requires the described object to support its function, delivery or coherence.

	isRequiredBy
Namespace	dcterms
Datatype	String
Method of input	Manual
Example (gui)	("http://www.myslides.com/myslides.ppt") *note: Where the image being described is required for an online show*
Schema	"http://www.myslides.com/myslides.ppt"

Table 54: ESE-CHO Is Required By element overview.

4.4.20. Requires

(The opposite of *Is Required By*), a related object that is required by the described object to support its function, delivery or coherence.

	requires
Namespace	dcterms
Datatype	String
Method of input	Manual
Example (gui)	("http://ads.ahds.ac.uk/project/userinfo/css/oldbrowser.css") *note: Where the resource described is an HTML file at http://ads.ahds.ac.uk/project/userinfo/digitalTextArchiving.html*
Schema	"http://ads.ahds.ac.uk/project/userinfo/css/oldbrowser.css"

Table 55: ESE-CHO Requires element overview.

4.5. Elements supplied by Europeana for a CH Object

The following elements are generated by Europeana during the ingestion phase. They are provided below only for completeness purposes:

4.5.1.Europeana language

This element is used to support the language facet in the Europeana.eu portal. As part of the ingest process the language(s) stated in element *Language* (if any) will be normalised and entered in this element.

4.5.2.Europeana URI

This element supports the internal functioning of the Europeana system. The value in this element is a unique identifier for each object record in the system. It is generated algorithmically based on an element in the source metadata that provides a unique identifier for the object.

4.5.3.Europeana year

This element is used to support the Timeline and the Date facet in the Europeana.eu portal. The value in this element is a four digit year (YYYY) from the Gregorian calendar. It is generated during the normalisation process from the value provided in the *Date* element (or one of the date refinements as appropriate) by the provider. The value supplied in the element *Date* may not be in this standard form but the normalisation process will attempt to identify a four digit year from the value supplied. Ideally therefore the value in the element *Date* should contain a year in the form YYYY. Objects where no europeana:year value can be generated will not appear in the Timeline or Date facet.

4.6. Elements supplied for a collection of CH Objects

This category provides metadata information for logical groupings of contributed CH objects within a museum.

4.6.1. Title

A mandatory element that keeps the title of the collection.

	title
Namespace	dc
Datatype	LangString
Method of input	Manual
Example (gui)	(“en”, “TNHM mineral collection images”) (“et”, “Mineraalid Eesti Loodusmuuseumi kogudest”)
Schema	“TNHM mineral collection images”@en “Mineraalid Eesti Loodusmuuseumi kogudest”@et

Table 56: ESE-CHO Collection Title element overview.

4.6.2. Creator

A mandatory element that refers to the name of the creator or creators of the collection. Map each name to a separate, repeated creator element if possible. Ideally choose a preferred form of name from an authority source. If you do not use an authority source, use a consistent form of the name e.g. Shakespeare, William.

	creator
Namespace	dc
Datatype	LangString
Method of input	Manual
Example (gui)	(“Rutt Hints”)
Schema	“Rutt Hints”

Table 57: ESE-CHO Collection Creator element overview.

4.6.3. Subject

This mandatory element refers to the subject(s) of the collection. This can include topics, people and places but consider using the spatial and temporal elements for places and time periods if the source data allows. Best practice is to use a separate instance of the element for individual subject terms.

	subject
Namespace	dc
Datatype	LangString
Method of input	Manual
Example (gui)	(“en”, “Earth science”) (”et”, “ Geoteadused”) (“en”, “Geology”) (“et”, “ Geoloogia”) (“en”, “Mineralogy”) (“et”, “Mineraloogia”)
Schema	“Earth science”@en “ Geoteadused”@et “Geology”@en

	"Geoloogia"@et "Mineralogy"@en "Mineraloogia"@et

Table 58: ESE-CHO Collection Subject element overview.

4.6.4. Description

A prose description of the collection, elaborating on the information in the metadata. This is an optional data element.

	description
Namespace	dc
Datatype	LangString
Method of input	Manual
Example (gui)	("en", " geological Collections of Estonian Museum of Natural History") ("et", "Mineraalide pilte Eesti Loodusmuuseumi geoloogilistest kogudest") ("et", "Mineraloogia")
Schema	"geological Collections of Estonian Museum of Natural History"@en "Mineraalide pilte Eesti Loodusmuuseumi geoloogilistest kogudest"@et "Mineraloogia"@et

Table 59: ESE-CHO Collection Description element overview.

4.6.5. Contributor

This is an optional data element that keeps the name of contributors to the collection. Names can include those of people, organisations or services. Map each name to a separate repeated contributor element if possible. Ideally choose a preferred form of name from an authority source. If you do not use an authority source, use a consistent form of the name e.g. Shakespeare, William. For each contributor the kind (role) of contribution has to be provided. A number of roles are provided as vocabulary terms in the MARC Code List for Relators (http://id.loc.gov/vocabulary/relators.html).

	contributor
Namespace	naturaleurope
Datatype	LangString
Method of input	Manual
Example (gui)	(“Friedrich Schmidt (collector)”) (“Tiit Hunt (photographer)”)
Schema	“Friedrich Schmidt”, “Collector”, “roles:http://id.loc.gov/vocabulary/relators/col” “Tiit Hunt”, “photographer”, “roles:http://id.loc.gov/vocabulary/relators/pht”

Table 60: ESE-CHO Collection Contributor element overview.

4.6.6. Type

This is an optional data element that keeps the type of the collection as recorded by the content holder. Type includes terms describing general categories, functions, genres, or aggregation levels for content. This should be used to record the values given in the source data which, ideally, will have been taken from a controlled vocabulary. A controlled DC Type vocabulary is provided (available at http://dublincore.org/documents/dcmi-type-vocabulary/).

	type
Namespace	naturaleurope
Datatype	LangString or VocabularyTerm
Method of input	Manual, preferably the values should be taken from the provided vocabulary.
Example (gui)	(“en”, “subfossil”) (“et”, “subfossiil”) or (“Image”)
Schema	“subfossil” @en

	"subfossiil"@et or "Image", "dcmitype:http://purl.org/dc/dcmitype/Image"

Table 61: ESE-CHO Collection Type element overview.

4.6.7. Identifier

This optional element can be used for an identifier of the collection.

	identifier
Namespace	dc
Datatype	String
Method of input	Manual
Example (gui)	("http://nla.gov.au/nla.pic-an7678346-1-v-cd")
Schema	"http://nla.gov.au/nla.pic-an7678346-1-v-cd"

Table 62: ESE-CHO Collection Identifier element overview.

4.6.8. Coverage

Coverage is an optional element that can be used for either spatial or temporal aspects of the collection. Values will typically include either a spatial location (place name or geographic co-ordinates) or a temporal period (a date range or period label).

	coverage
Namespace	dc
Datatype	LangString
Method of input	Manual

Example (gui)	(“en”, “Boston, MA”) (“en”, “1995-1996”)
Schema	“Boston, MA”@en “1995-1996”@en

Table 63: ESE-CHO Collection Coverage element overview.

Chapter 5

System Architecture

EuroMuse is an application used by museum curators for publishing CHOs, describing CHOs and CHO Collections, as well as for managing their metadata. This chapter describes the complete system's architecture, identifies its basic components and provides an in depth analysis of the internal functionality. Built as a web application, the system adopts the Google Web Toolkit (GWT) [19] framework which facilitates the development of web applications as desktop applications, performing business logic operations on server side, as well as on client side. The Client Side logic operates within the web browser running on a user's local computer, while the Server Side logic operates on the web server hosting the application.

For the development of the application we adopted several design patterns, which are presented in detail in the following sections. Model-View-Presenter and Observer were used on the client side, and a multi-tier architecture was implemented on the server side. Figure 9 displays the overall system architecture. The analysis of the architecture has been broken into two parts; Section 5.1 describes the Client Side architecture, and Section 5.2 describes the Server Side architecture.

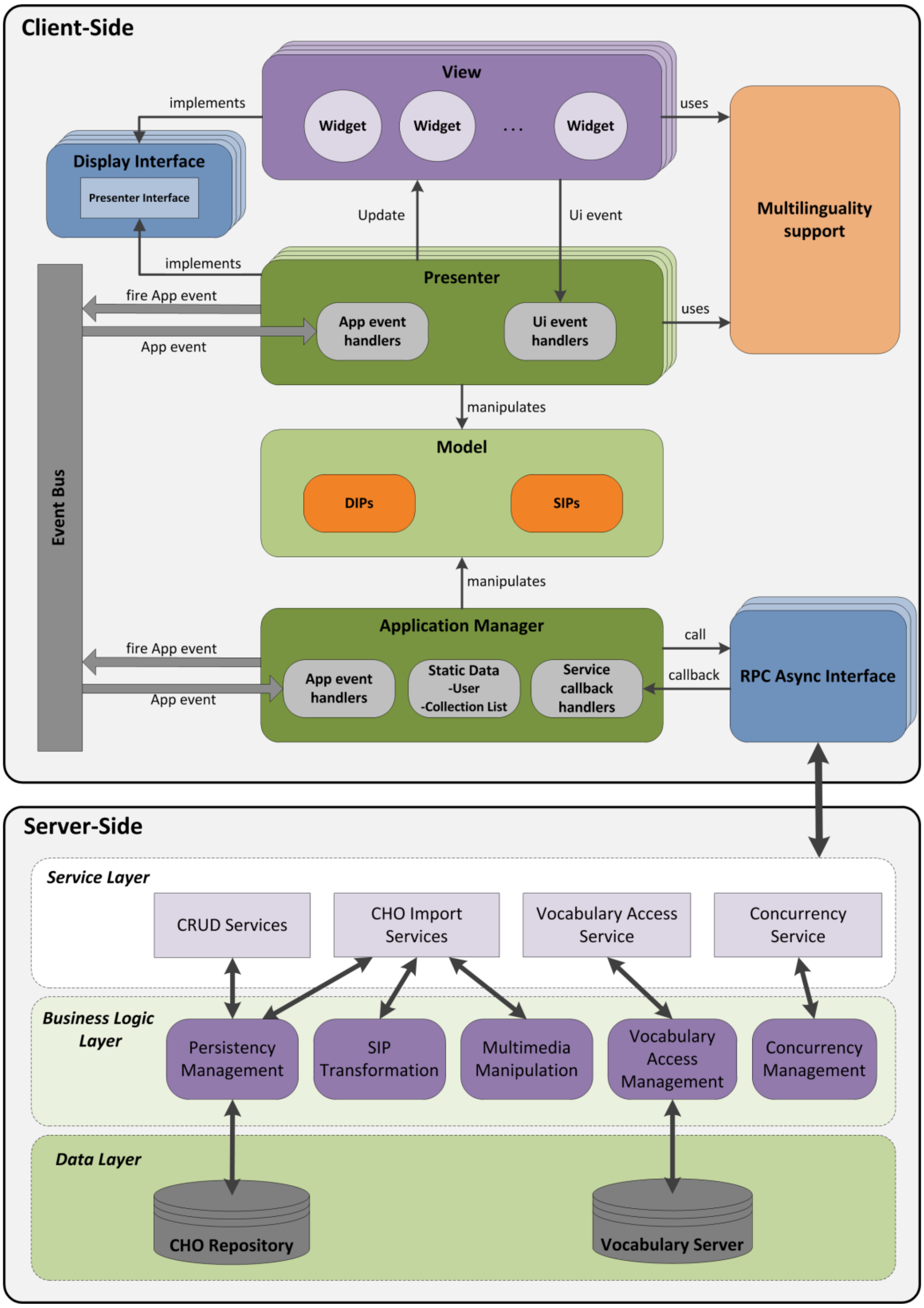


Figure 9 : Overall system architecture, containing client-side (top) and server-side (bottom) along with the included modules.

5.1. Client Side

The Client Side of the application is responsible for the interaction with the user. All the actions performed by an individual using the system, are handled by the client side logic, which undertakes the presentation of the information as well as the communication with the server when needed. To achieve the scalability of the User Interface, and the distinction between the components forming the client logic, a number of well-established patterns were adopted.

The design pattern we adopted for defining the system's client side architecture is the Model-View-Presenter (MVP) [27] design pattern, which is described in Section 5.1.1. Along with MVP, we implemented the Observer pattern using an Event Bus (Section 5.1.2), and the Mediator pattern using the Application Manager (Section 5.1.3).

5.1.1. Model-View-Presenter (MVP)

Most screens in a Web application contain controls that allow the users to review application domain data. A user can modify the data and submit the changes. The client logic retrieves the domain data from the server, handles user events, alters other controls on the page in response to the events, and submits the changed domain data back to the server. Including the logic behind these functions in the Web page makes the code complex, difficult to maintain, and hard to test. In addition, it is difficult to share code between Web pages that require the same behavior.

This pushes the need for an architectural design that:

- Maximizes the code that can be tested with automation. (Web pages containing HTML elements are hard to test.)
- Code sharing between pages that require the same behavior.
- Separation of Business logic from User Interface logic to make the code easier to understand and maintain.

These requirements drove the creation of the Model-View-Presenter (MVP) Design Pattern [27]. MVP introduces the separation of the responsibilities for the visual display and the event handling behavior into different entities named, respectively, the view and the presenter.

MVP was created mainly for the development of desktop applications, due to the difficulty to write and debug complex business logic code in JavaScript (used in web applications for programming the client's browser), but the usage of GWT allowed us to write highly complex logic code and thus implement the MVP on the client side of our application.

The components in the system architecture that compose the MVP pattern are: a) the Model, b) the View, c) the Presenter and d) the Display Interface. Figure 10 presents the interaction between these components. The View is responsible for the user interface. It implements the Display Interface (green circle) whose members are technology striped UI elements of the view. The Presenter controls the behavior of the view using the Display Interface. The user interacts with the view and all the user actions generate a User Interface event. When such an event occurs on the view, it is forwarded to the presenter whose responsibility is to handle the event. Presenter is manipulating view by talking to interface representation of the view and should never reference directly view members (UI controls). For his logic operations the Presenter uses the Model, which represents the business objects.

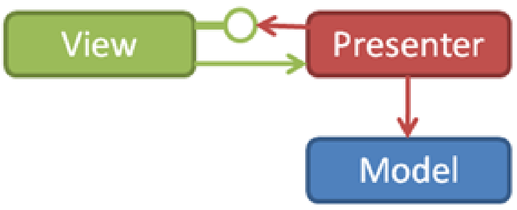


Figure 10: Model-View-Presenter (MVP) Design Pattern.

Apart from MVP, a similar pattern used in many web applications is Model-View-Controller (MVC) [28]. Figure 11 presents the diagram of this pattern which is composed of the View, the Controller and the Model. The View in this pattern is also the component responsible for the presentation and interaction with the user. Each action of the user produces an event which is handled by the Controller, who converts the event into an appropriate user action understandable for the model. The Controller notifies the Model of this action, possibly resulting in a change in the model's state. Subsequently, the view queries the model and generates the appropriate interface.

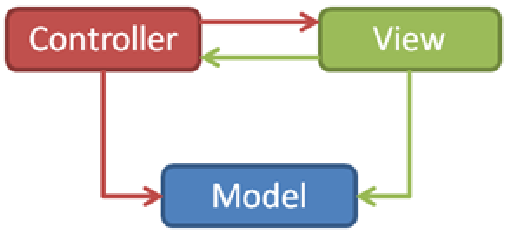


Figure 11: Model-View-Controller (MVC) Design Pattern.

The two patterns are very similar, but yet a lot of differences derive after careful observation. MVC is mostly used in web applications where it dominates over MVP. On the other side, MVP has many advantages over MVC in desktop applications. Although EuroMuse is a web application, the evolution of the Web has filled the gap between desktop and web applications and introduced the Rich Internet Applications (RIAs). RIAs are web applications with many characteristics of desktop applications. EuroMuse is developed as a RIA and for this reason we chose to follow the MVP pattern.

The advantages of MVP over MVC in desktop/RIA applications are:

- In MVP the View is completely unaware of the Model and the Presenter is the one exclusively interacting/manipulating it, whereas in MVC both the View and the Controller are aware of the Model. This makes MVP easier to maintain when the Model undergoes modifications because only the Presenter needs to be updated.
- In MVP the communication between the View and the Presenter is done through the Display Interface, thus allowing the independent development of the logic (Presenter) and the user interface (View) by the developers and the designers respectively, after the Display Interface has been defined.
- The usage of the Display Interface in MVP facilitates the unit testing of the View and the Presenter, by allowing the fast creation of mock implementations.

5.1.1.1. Model

The Model encompasses all business objects used by the system. These include the CHOs, the Metadata concerning the CHOs, as well as the metadata concerning the CHO Collections and the details of the users in the system. Due to the fact that the Persistency Management Module (see Section 5.2.2.1) is built as an Open Archival Information System (OAIS) [29], there is a need for organizing the model into Information Packages.

An **Information Package (IP)** refers to the content information and associated preservation description information, which is needed to aid in the preservation of the content information. IPs are further split into **Dissemination Information Packages (DIPs)** and **Submission Information Packages (SIPs)**. Dissemination Information Package refers to Information Packages that are to be exported (disseminated) by an Open Archival Information System, whereas Submission Information Package refers to an Information Package submitted to an OAIS according to the OAIS Reference Model.

When the system needs to present information about a business object, the server side services are used to transfer the DIPs to the client side. When an update on the Model needs to take place, the client side sends the SIPs with the new information to the server side.

5.1.1.2. View

In the MVP pattern, the view is responsible for all the information presentation (colors, borders, UI layout) and the management of the controls on a page or part of a page. All HTML and the StyleSheets are handled by views.

In our system, each view is a composite widget, aggregating a set of simple widgets. Simple widgets refer to the simple HTML elements (e.g. tables, labels, buttons, textboxes, forms, images etc.) The views dynamically create and manipulate these widgets, according to the system actions, or use them to capture user actions. Moreover, views have no notion of the model. That is to say a view doesn't know what it is displaying; it simply knows that it has, for example, 3 labels, 3 textboxes, and 2 buttons that are organized in a vertical fashion.

Each view is coupled with one presenter. The presenter instantiates the view, controls it throughout its lifecycle and destroys it when the time comes. During instantiation, the presenter instructs the view where to “sit” on the screen. This way a single web page can be composed by multiple presenters manipulations various views.

When the user interacts with the web page, a browser *event* takes place. This event can be traced using GWT mechanisms, and handled whichever way needed. The important events are handled by the views, or delegated to the corresponding presenters, if the presenter has asked to be informed about the specific type of event. In turn, the presenter might instruct the view to update some of its visual parts, based on the event. For example, if the user clicks the delete button next to an object in a list, the view delegated the click event to the presenter. The presenter informs the server to delete the object, and then instructs the view to remove the object from the list.

5.1.1.3. Presenter

A Presenter contains all of the client side logic of our application. It is the key in MVP design pattern, for it is a bridge between the Model and the View. As a general rule, for every View there is a presenter that instantiates and manipulates it, but there might be cases where one presenter needs to control multiple views. The presenter has handlers for the events that take place in his view/s or in the application in general (see Event Bus in Section 5.1.2). The lifecycle of a presenter is composed of four phases.

- **Initialization phase:** The presenter initializes its corresponding view/s, and instructs them where to deploy. In addition, the presenter registers handlers to the crucial UI events and application events. When the initialization and handler registration finish, the presenter goes into the sleeping phase.
- **Sleeping phase:** This is the phase where the presenter does nothing but wait for a UI event from the view, or an application event.
- **Operating phase:** When an already registered event is raised, the presenter wakes up and handles it (using its UI/App Event Handlers) by contacting the server, updating the view or even raising new events.
- **Termination phase:** When the user leaves the application, the presenter goes into the termination phase, before being destroyed. During this phase, the presenter might need to inform the server for the termination.

The communication of the presenter with the services residing on the server, takes place through the Application Manager who acts as a Mediator. More on this subject can be found in Section 5.1.3.

As already stated, the presenter has no knowledge of any widget-based code inside the view and all the interaction with the view is done through the Display Interface.

5.1.1.4. Display Interface

In an MVP architecture, the Presenter and its corresponding View must be lightly coupled, to facilitate unit testing. This goal is achieved by utilizing Display Interfaces between the

Presenters and the Views. Each View implements a Display Interface, which defines all the methods needed by the presenter for the manipulation of his view. In turn, the presenter has a reference to the view interface instead on the concrete implementation of the view. By doing this, the real view can be easily replaced with a mock implementation to run tests.

In various cases, it is crucial for a view to notify the presenter. This is done through the Presenter Interface contained inside the Display Interface, which is implemented by the presenters.

5.1.2. Event Bus

The use of MVP allows the development of multiple presenters on the same screen of the application, providing better control over the different parts of the interface. This creates the need for some kind of interaction between them. The easiest way to achieve this is by letting each presenter know about the existence as well as the structure of the other presenters. The specific implementation creates highly coupled presenters, which makes unit testing almost impossible.

To avoid the coupling between the presenters, we decided to use the Observer pattern (subset of the Publish/Subscribe pattern) with the usage of an Event Bus. The Event Bus is a mechanism for a) passing events and b) registering to be notified of some subset of these events. Respectively, presenters can register to be notified when some events occur (App Event Handlers), or fire events on the Event Bus to notify other presenters.

Using this pattern, the presenters can be developed completely uncoupled by processing all the communication between them through the Event Bus. This makes the presenters completely unaware of each other, which eases unit testing. The level of coupling between the presenters can vary depending on the needs of the application, and is up to the developers to decide.

It's important to keep in mind that not all events are to be placed on the Event Bus. Blindly dumping all of the possible events on the Event Bus can lead to a chatty application that can get bogged down in event handling. Specifically, the UI events handled by the presenters should not be forwarded to the Event Bus because they contain a lot of unneeded data, but instead another Event should be instantiated and fired, containing just the information needed.

As an example, when the user clicks on a button contained in a view, if no other presenters need to be informed of the click, the presenter should not pass the event to the Event Bus as it will not have any registered handlers. Moreover, when other presenters need to be informed (e.g. for the click on the “Save” button) the presenter handling the UI event should not pass it as it is to the Event Bus because it contains a lot of data about the coordinates of the mouse, reference to the button etc. which have no use to the other presenters listening to the event. Instead, the presenter should instantiate a new event with just the needed data and pass it to the Event Bus.

App-wide events are really the only events that we passed around on the Event Bus. The app is uninterested in events such as "the user clicked enter" or "an RPC is about to be made". Instead, we pass around events such as a record being updated, the user selects a record to view its description, or an RPC that deleted a record has successfully returned from the server.

5.1.3. Application Manager

The Application Manager module has been implemented as a Mediator of the Client Side, residing between the Presenters and the Server Side, making calls to the services exposed in the Service Layer, and notifying the Presenters for the service response. In addition, the Application Manager holds some static data needed throughout the application (e.g. the user details, etc.).

When a presenter needs to communicate with the Server Side it places an App event (Application Event) to Event Bus. The Application Manager handles this event using his App Event Handlers and communicates with the Server Side via RPC Async Interface. When this service call has been successfully executed on the server and the response is received (by the Service Callback Handlers), the Application Manager places an App event on the event bus to inform the presenter of the outcome.

The Application Manager has also been developed with another important aspect in mind, client side caching. Although the application has no cache on the client side yet, the architecture allows for a cache to be created next to the Application Manager, which he can access before requesting data from the server. This can drastically improve the speed of the system.

5.1.4. Multilinguality support

Developing applications that can be used in different countries, with different languages, often requires applying specific techniques. The usage of GWT though, simplifies dealing with internationalization matters.

The multilinguality support module uses the GWT approach for internationalization matters. Views, Presenters and Application Manager use this module in order to take the appropriate data for each language supported and display it to the user interface. GWT provides i18n support with its Constants interface allowing you to work with string literals in different languages, and with the Messages interface, which adds singular/plural considerations.

For the development of our system, we implemented the static string internationalization technique. Static string internationalization is the most efficient way to localize our application for different locales in terms of runtime performance. Also, our application has only constants and parameterized messages in its user interface, so this technique could be the best choice.

This approach is called "static" because it refers to creating tags that are matched up with human readable strings at compile time. At compile time, mappings between tags and strings are created for all languages defined in the module. The module startup sequence maps the appropriate implementation based on the locale setting using deferred binding.

For more detail in GWT's i18n Support, see Section 2.10.4.

5.1.5. RPC Async Interface

The RPC mechanism handles the communication between the client-side and the server-side. It uses GWT's mechanisms, which allow fast deployment of remote services, eliminating the need to create Java Servlets. For more information, see Section 2.10.3.

5.2. Server Side

The Server Side part of our application follows a multi-layered Architectural pattern consisting of three basic layers: Service Layer, Business Logic Layer and Data Layer.

The advantages of this approach include the increased system maintainability, reusability of the system components, scalability, robustness, and security. Moreover, the adoption of a multitier architecture and the provision of well-defined interfaces for each layer, allows any of the layers to be upgraded or replaced independently (with minimum effort) as requirements or technology change.

5.2.1. Service Layer

The *Service Layer* controls the communication between the client logic and the server logic, by exposing a set of services (operations) to the client side components. These services actually comprise the middleware concealing the application's business logic from the client. The basic system services are listed below:

- *CRUD Service:* Facilitates the creation, retrieval, update and deletion of a CHO, a CHO Metadata record, a CHO Collection, a user etc.
- *CHO Import Services:* Supports the importing of record-based xml metadata in order to be converted to SIPs using the SIP Transformation Module, as well as the import of media files which are published and the metadata information that they hold is extracted and used for the description of the new CHO. The SIPs are then transferred to the Persistency Management Module for storage into the CHO Repository.
- *Vocabulary Access Service:* Responsible for the access to the taxonomic terms and vocabulary.

- *Concurrency Service:* Provides the basic methods for acquiring/releasing/refreshing locks on a CHO Metadata Record or a CHO Collection.

5.2.2. Business Logic Layer

The *Business Logic Layer*, also known as Domain Layer, contains the business logic of the application and separates it from the Data Layer and the Service Layer, which is used by the Client Side's modules. It consists of four basic modules (analyzed in detail at the end of this section):

- *Persistency Management:* Manages operations requiring the submission/fetching of information packages to/from the CHO Repository.
- *SIP Transformation:* Used for the transformation of record-based XML metadata to SIPs, using mappings produced from the Europeana SIP Creator Tool [30].
- *Multimedia Manipulation:* Used during the importing of media objects. Extracts metadata from files in supported formats, and creates CHO metadata records with this information.
- *Vocabulary Access Management:* Provides access to indexed vocabularies or authority files residing on a Vocabulary Server. Such vocabularies may refer to the taxonomic classification of a CHO, to publicly sourced authority files of persons, places, etc.
- *Concurrency Management:* Applies a pessimistic locking strategy to CHO Metadata or CHO Collection Metadata records in order to overcome problems related to the concurrent editing by multiple users.

Since the modules of the Business Logic Layer contain the main logic residing on the Server Side, they are described below in further detail.

5.2.2.1. Persistency Management Module

The Persistency Management Module is responsible for the communication between the application and the CHO Repository. Whenever a system service needs to perform an update on the repository, the module compiles the input information into SIPs and uses the CHO Repository API to initiate the ingest phase. Accordingly, the module retrieves DIP packages from the CHO Repository and promotes them to the service layer when needed. The Client Side logic interacts with this module through the service layer when a user reviews/updates a CHO/CHO Collection, etc.

5.2.2.2. SIP Transformation Module

This module provides an *Application Programming Interface (API)* containing methods used for the transformation of record-based XML metadata to SIPs, using mappings compliant with the format produced and used by the Europeana SIP Creator Tool. It actually serves the requirement of metadata unification, by supporting the migration of already existing NHM legacy metadata to SIP compliant records that will be transferred for storage, in a later step, to the CHO Repository. The Client Side logic interacts with this module through the service layer when a user imports.

5.2.2.3. Multimedia Manipulation Module

The Multimedia Manipulation Module is responsible for the extraction of the metadata information embedded in media files. The CHO Import Services uses this module during the import of Media Objects, in order to automatically enrich the metadata description of the new CHOs with everything that comes along the file.

5.2.2.4. Vocabulary Access Management Module

The Vocabulary Access Management Module is responsible for the indexing and accessing of vocabularies or authority files residing on a Vocabulary Server. Such vocabularies may refer to the taxonomic classification of a CHO, to publicly sourced authority files of persons, places, etc.

The first release of EuroMuse uses the Catalogue of life. The Catalogue of Life [31], started in June 2001 by Species 2000 and Integrated Taxonomic Information System (ITIS), is planned to become a comprehensive catalogue of all known species of organisms on Earth.

The present Catalogue is compiled with sectors provided by 99 taxonomic databases from around the world. Many of these contain taxonomic data and opinions from extensive networks of specialists, so that the complete work contains contributions from more than 3,000 specialists from throughout the taxonomic profession. Species 2000 and ITIS teams peer review databases, select appropriate sectors and integrate the sectors into a single coherent catalogue with a single hierarchical classification. It is planned to introduce alternative taxonomic treatments and alternative classifications, but an important feature is that for those users who wish to use it, a single preferred catalogue, based on peer reviews, will continue to be provided.

Biological classification

Biological classification, or *scientific classification* in biology, is a method by which biologists group and categorize organisms by biological type, such as genus or species. Biological classification is part of scientific taxonomy.

In biological classification, rank is the level (the relative position) in a taxonomic hierarchy. Ranks are assigned based on subjective dissimilarity, and do not fully reflect the gradational nature of variation within nature. In most cases, higher taxonomic groupings arise further back in time: not because the rate of diversification was higher in the past, but because each subsequent diversification event results in an increase of diversity and thus increases the taxonomic rank assigned by present-day taxonomists.

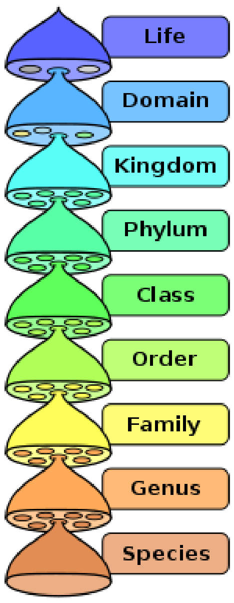


Figure 12: The hierarchy of biological classification's eight major taxonomic ranks.

The Catalogue of life website provides web services for external usage of their taxonomy. They also provide the full database of their system, which is regularly released for free download. Using D2R Server [32], the relational database was published in RDF format, with SPARQL support. The RDF data format was designed to comply with the Simple Knowledge Organization System (SKOS) vocabulary. An example of the published data can be seen in Figure 13.

The taxonomy expressed in SKOS vocabulary format facilitates the semantic linkage of museum objects to the catalogue of life taxonomic classification nodes. This alone provides effective mechanisms for searching similar objects not only in the context of a single museum, but in a global scope. The SKOS data will be hosted in the central Natural Europe federation node in order to provide centralized access to all instances of EuroMuse.

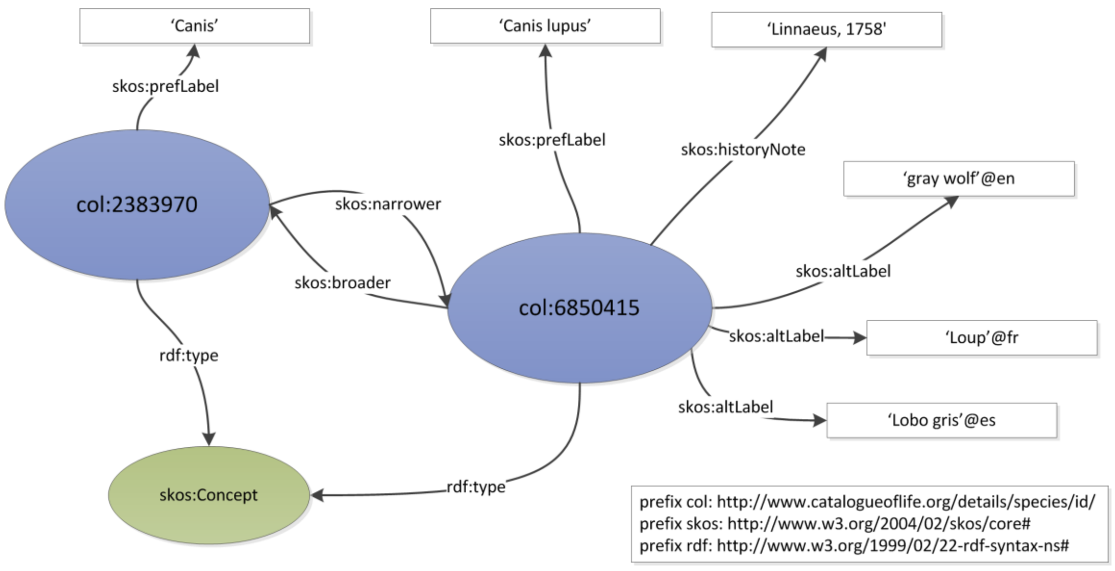


Figure 13: SKOS example.

Besides the SKOS representation, the taxonomy has also been indexed with the Solr search platform, in order to provide faster searching of terms inside the taxonomy. To achieve the indexing of the data, the database had to undergo some transformation, and the final form of each of the species includes the names of all ranks in higher order inside the tree.

```
<doc>
      <str name="col_author">Linnaeus, 1758</str>
      <str name="col_class">mammalia</str>
      <str name="col_family">canidae</str>
      <str name="col_genus">canis</str>
      <str name="col_kingdom">animalia</str>
      <str name="col_name">canis lupus Linnaeus, 1758</str>
      <str name="col_order">carnivora</str>
      <str name="col_phylum">chordata</str>
      <int name="col_rank_id">8</int>
      <str name="col_rank_name">species</str>
      <str name="col_species">lupus</str>
      <str name="id">6850415</str>
</doc>
```

Example 5.1: XML example of a simple species inside solr.

The curators using EuroMuse are provided with the ability to use the catalogue of life taxonomy during the enrichment of the object metadata.

5.2.2.5. Concurrency Management Module

The Concurrency Management Module controls the locking/unlocking mechanisms implemented to overcome the problems arising when two people edit the same data at the same time on the web. Before describing our approach, we will define the problem and the recommended solutions.

Although the usual method of ensuring the concurrency of an application is to use the locking mechanisms, which are pre-build in most databases, we decided to move the mechanism higher in our application logic and thus implementing this module. This was mostly due to the use of exist-db [32] for the storage of the data. Exist-db is one of the most powerful native XML Databases but the transaction mechanisms provided along are yet in very primitive state and are currently limited only to the functionality needed for internal crash recovery.

EuroMuse provides the user with a basic interface that facilitates the creation of new objects/collections, retrieval of existing objects/collections from the database storage, update or deletion of the data.

These kinds of services are often described as CRUD (Create, Read, Update and Delete) services. An example would be a user creating and then updating the information about a CHO in the system. The CRUD operations for each action would be:

1. Create – create a new object in the system
2. Read – browsing through the collections and viewing the object
3. Update – changing the information about the object and save the updated object
4. Delete – deleting the object

In this above example, only one person is updating the object, so the object in the database is the same between the Read and Update steps. However, one of the great advantages that wed applications offer is that multiple users can use the system at the same time, allowing the collaboration, but also producing some unexpected results if these operations are uncontrolled. The common problems that occur in these cases are: the lost update problem, the dirty read problem and the incorrect summary problem.

Lost Update Problem

The lost update is the most common problem and in order to illustrate, we will use an example.

Consider two colleges in a museum, Joe and Bob. Using our system, Joe wants to update some fields about the “Monachus” object. He opens his browser, logs in the system, browses through the collections until he finds the object and makes some changes. Then Joe decides he wants to get some coffee, before completing the changes and saving the record.

In the meantime, Bob who goes through all the objects one by one adding just an internal identification number of the museum, reaches “Monachus”. At this point, the data displayed to Bob doesn’t contain the changes that Joe already did. Bob updates the data, clicks the “Save” button and moves to the next record.

Joe comes back with his coffee. At this point, Joe's view of "Monachus" data is not consistent with the data in the database. Bob's change in the data is not shown in Joe's web browser. Most web applications operate in this manner – they only send data to the user (Joe) when the user requests it. Now Joe makes more changes to the object and clicks the "Save" button. This causes Bob's update to be lost.

Bob gets back from lunch, searches for and views "Monachus" object again, and sees that the internal identification number of the museum is missing. Bob is angered and fires off an email to his boss telling him the new museum web application is broken and he wants to use the old application.

As you can imagine, this is not good for the application development team or the salespeople. Figure 14 illustrates the sequence of events leading up to the lost update.

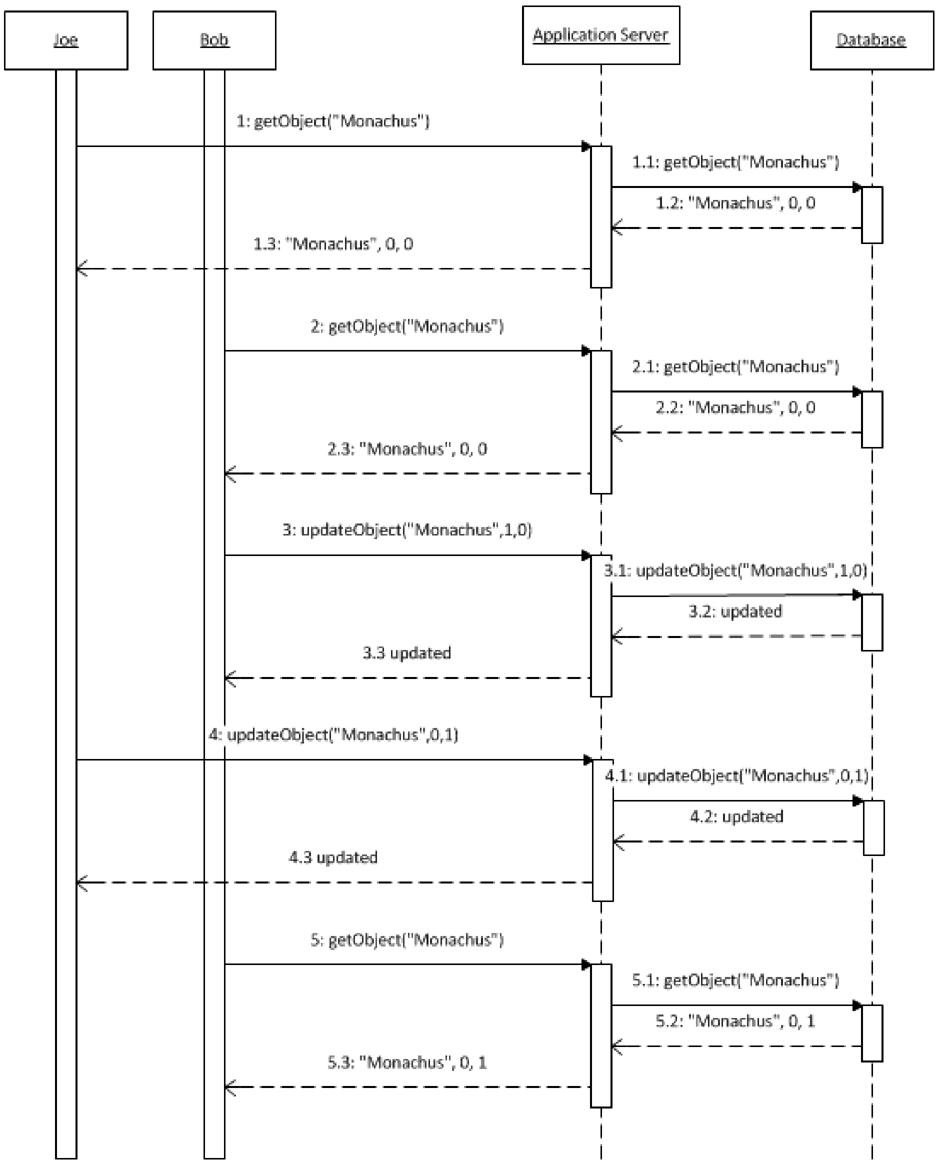


Figure 14: The lost update problem.

Incorrect Summary Problem

The incorrect summary problem occurs when one transaction takes a summary over the values of all the instances of a repeated data-item, while a second transaction updates some instances of that data-item. The resulting summary does not reflect a correct result for any (usually needed for correctness) precedence order between the two transactions (if one is executed before the other), but rather some random result, depending on the timing of the updates, and whether certain update results have been included in the summary or not.

Dirty read problem

The dirty read problem occurs when transactions read a value written by a transaction that has been later aborted. This value disappears from the database upon abort, and should not have been read by any transaction ("dirty read"). The reading transactions end with incorrect results.

To handle the concurrency problems, two main concurrency control mechanisms are proposed: optimistic and pessimistic. In the next paragraphs we will try to explain in depth these two mechanisms, and then describe the exact model we used, which is based on the pessimistic mechanism.

Optimistic Concurrency Control

Optimistic concurrency control (OCC) is a concurrency control method that assumes that multiple transactions can complete without affecting each other and that therefore transactions can proceed without locking the data resources that they affect. Before committing, each transaction verifies that no other transaction has modified its data. If the check reveals conflicting modifications, the committing transaction rolls back.

When using OCC, a timestamp of the database record is send with the data when the user retrieves a record. The same timestamp is send back to the server, when the user updates the record. Then the server uses this timestamp to determine if someone else changed the data in the database after this user last read it. If the data in the database was not changed the user's update succeeds, otherwise it fails and the user gets a friendly error message along the lines of *"Another user has changed the data since your last request. Please try again"*. At this point, some implementations might try to help the user merge their changes with the current state in the database. Other implementations send the user the latest state of the data in the database and force them to re-enter their changes. As you can imagine, optimistic concurrency control can get very frustrating if you have a large web form and get many "please try again" messages.

Going back to our example, if optimistic concurrency control was used, both Joe and Bob would get the same timestamp value from the database in their initial read. Bob's update would succeed because the timestamp matches the current timestamp in the row for the object. Besides changing the phone number, Bob's change causes the database to update the object's timestamp column to match the time Bob submitted his change.

Now Joe sends his address change with the old timestamp and the application would reject it because the timestamp in the database is not the same as the timestamp that Joe sends with the update. The application would give Joe an error message along with the current state of Bob's record, including the new timestamp. Joe now has to redo all his changes to the object.

OCC is generally used in environments with low data contention. When conflicts are rare, transactions can complete without the expense of managing locks and without having transactions wait for other transactions' locks to clear, leading to higher throughput than other concurrency control methods. However, if conflicts happen often, the cost of repeatedly restarting transactions hurts performance significantly; other concurrency control methods have better performance under these conditions. [33]

Pessimistic Concurrency Control

Pessimistic concurrency control introduces the concept of locking data to prevent others from attempting to modify it. A user can only change data if the user has its lock. The lock on the data is obtained when the data is initially read and is released when the user sends their changes to the server. If a user attempts to obtain the lock when another user already has it, it will fail and the user would get an error message. Depending on the lock mode (*shared*, *exclusive*, or *update*), other users might be able to read the data even though a lock has been placed.

- SHARED - allow multiple users to read data, but do not allow any users to change that data.
- EXCLUSIVE - allows only one user/connection to update a particular piece of data (insert, update, and delete). When one user has an exclusive lock on a row or table, no other lock of any type may be placed on it.
- UPDATE - no other user can read or update the row and ensures the current user can later update the row.

Locks can create a number of issues. If the user goes for a coffee break, the record remains locked, denying anyone else the ability to update the record, even if it has been untouched by the initial requestor. With pessimistic concurrency, some things to consider are how locks are acquired and released, if locks automatically expire after a period of time, and if a user can forcefully unlock data locked by another user.

If the example application used pessimistic concurrency control, Joe's request to read object's data from the database would give Joe the lock on "Monachus". Later, when Bob would read the object with the intent to modify it, Fred would be given a "This record is currently locked. Please try again later" application error since Joe already has the lock. Once Joe clicks the "Save" or "Cancel" button, the lock is released.

Our approach

Concurrency control in our system is based on the pessimistic mechanism, with some additions. The reasons that drove us to use the specific mechanism are:

2.1. The data in the system have high contention, which means there is a very high probability that two or more users will need to edit the same record at the same time.

2.2. The users of the system are museum workers, archeologists, etc., and most of the data they insert derives from a lot of research, so the optimistic mechanism could mean they would have to do part of their research again in case there is a conflict and their changes are lost.

To overcome these possible problems, we implemented a separate **concurrency component**, which was then used by Concurrency Management Module. The component has been designed to be independent of the model and the server logic.

As far as the user interface is concerned, we decided to introduce a different mode for the update process of an object, and gave users the ability to change between view and edit modes using a single push button. So when a user wants to view the object without making any changes, he doesn't have to acquire its lock. In the contrary, if he wants to edit it, when the mode change button is pressed, the system tries to acquire a lock on the object. If the lock succeeds, the mode is changed and the user can now edit anything he wants. When he finishes and updates the object, the lock is released. On the contrary, if another user already has a lock on the object, the system informs the user and the mode stays the same.

Besides the basic advantages of the pessimistic approach, there is the case someone stops the updating process without first releasing the lock. This produces the need for automatic lock expiration after a certain amount of time, which in turn requires the repeated refresh of the locks from the lock owner. Hence the system has been structured to release the lock 30'' (release threshold) after the last refresh, and the refresh of the lock takes place every 14'' (refresh threshold). The release threshold has been chosen so that users don't have to wait a lot of time for a lock to be auto-released, and the refresh threshold is set so two refresh service calls happen before the lock is released, to avoid releasing the call in the unlike event of a missed request to the server. Moreover, if the user owns the lock, but for some reason the browser window is refreshed or closed-reopened, the user is still able to re-acquire the lock (although any unsaved changes will be lost) because he is still the owner (unless the release threshold has passed and another user has acquired the lock).

Figure 15 shows the internal processing of the locks when a user tries to edit a record.

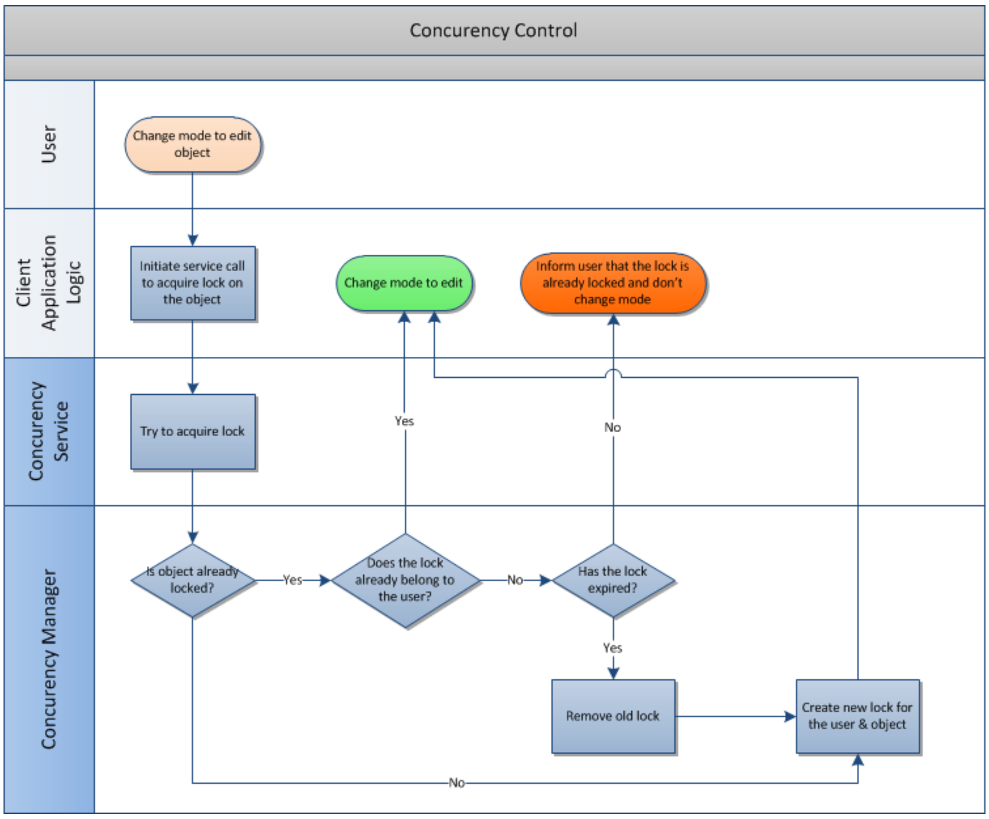

Concurrency Control
User
Client Application Logic
Concurency Service
Concurency Manager
Change mode to edit object
Initiate service call to acquire lock on the object
Change mode to edit
Inform user that the lock is already locked and don't change mode
Try to acquire lock
Yes
No
Is object already locked?
Yes
Does the lock already belong to the user?
No
Has the lock expired?
Yes
Remove old lock
Create new lock for the user & object
No

Figure 15: Concurrency Control Activity Diagram.

5.2.3. Data Layer

The *Data Layer* accommodates external systems that are used to store data accessed by this tool. Such systems are the CHO Repository, and the Vocabulary Server of the Natural Europe federal node.

5.2.3.1. CHO Repository

The CHO Repository holds all the data concerning the CHOs, CHO Collections and the Users of the system. It was developed by Konstantinos Makris, who provided us with the CHO Repository API which is used by the Persistency Manager to perform operations on the repository.

5.2.3.2. Vocabulary Server

The Vocabulary Server holds all the information about the vocabulary used by the system. It contains an instance of Lucene/Solr used by the system for the fast auto-complete features, and all the RDF/SKOS referring to the vocabulary.

Chapter 6

GUI Design Specification

In this chapter we are going to describe the strategy that we followed for the Graphic User Interface (GUI) design of our application. The user interface was designed very carefully in order to accomplish both the functionality of the application and the compliance with general design principles and guidelines.

The next subsections describe the general principles of the GUI design and the basic application structure.

6.1. Application Structure

This section describes the Graphical User Interface (GUI) of the application, by specifying its main graphical components. Figure 16 provides the basic application structure of the GUI having its basic graphical areas marked with different colors.

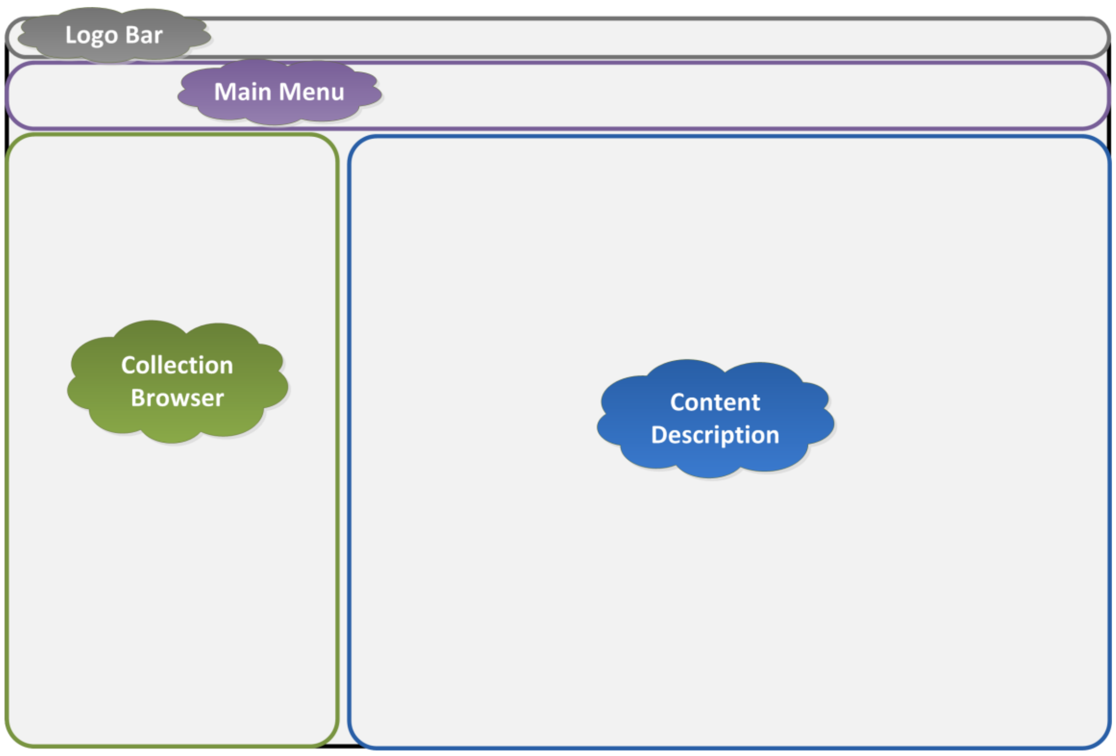


Figure 16 : Main GUI Specification.

Following Figure 16, the application GUI is separated in four basic graphical components summarized below:

- **Logo Bar:** Contains the emblem of the application to provide instant public recognition to the visitors, and the user interface language selection menu.
- **Main Menu:** Contains the basic menu items along with some information about the user's identity.
- **Collection Browser:** Facilitates the browsing through the CHO Collections and their CHO Metadata records.
- **Content Description:** Presents and manipulates both CHO Metadata and CHO Collection Metadata records. It actually forms a composite component, aggregating other smaller subcomponents presented in details in Figure 17 and Figure 18.

The active region of the application is the Content Description part of the GUI. This part alters according to the selected CHO/CHO Collection. If the content is CHO Collection Metadata (Figure 17), the Content Description consists of the following subcomponents:

- **Control Bar:** Contains the basic controls allowing the deletion of this CHO Collection, the saving of any changes, as well as the switch to Edit or View mode.
- **Content Box:** Hosts multiple values of a specific metadata element in a specific language.

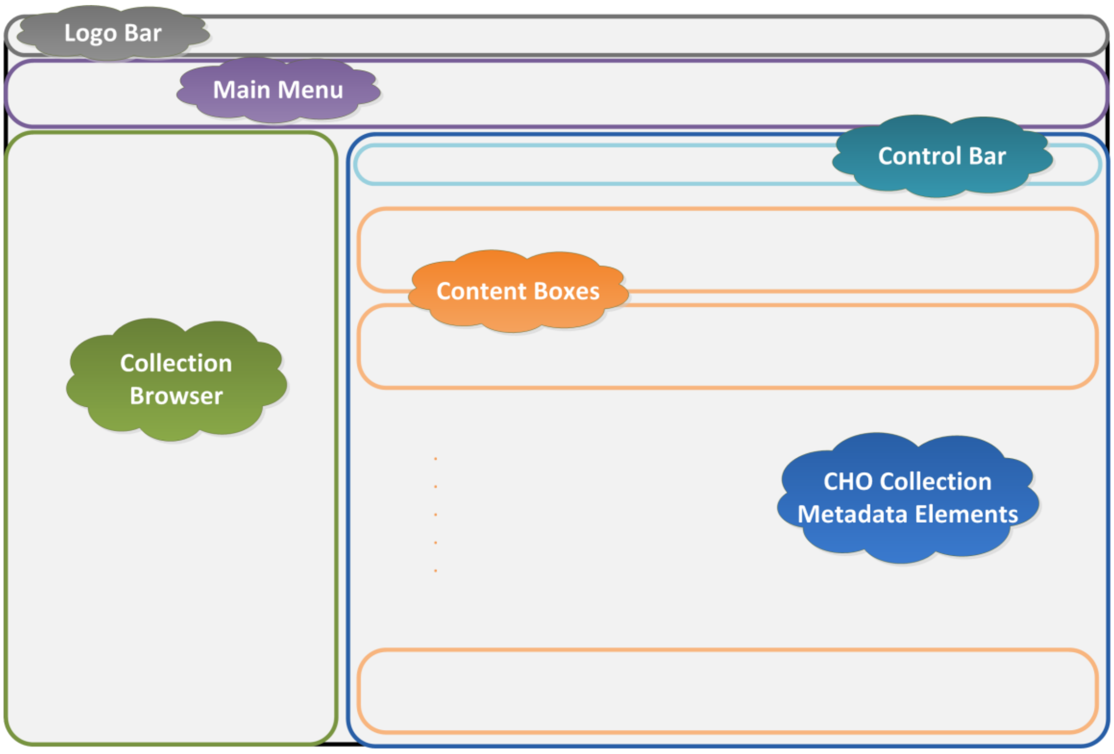


Figure 17 : GUI Specification of CHO Collection Metadata.

If the content is CHO Metadata (Figure 18), the Content Description consists of the following subcomponents:

- **Control Bar:** Contains the basic controls allowing the deletion of a CHO Metadata, the saving of any changes, as well as the switch to Edit or View mode.
- **CHO Preview:** Contains a small preview of the described CHO.
- **CHO Metadata Menu:** Used in order to browse different metadata categories.
- **CHO Metadata Elements:** Contains all the metadata element fields that can be used in order to describe a CHO.
- **Content Box:** hosts multiple values of a specific metadata element in a specific language.

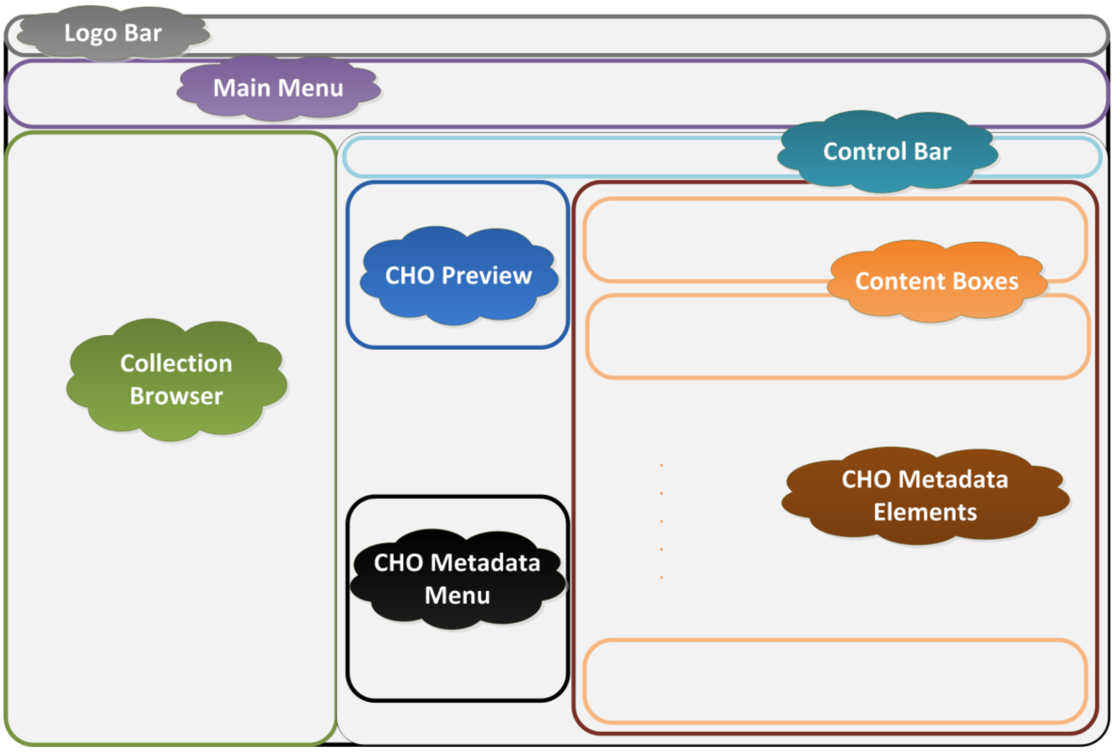


Figure 18 : GUI Specification of CHO Metadata.

6.2. Usability heuristics

In this section we will describe the ten usability heuristics according to Jacob Nielsen and how our application is in compliance with these recognized usability principles. They are called "heuristics" because they are more in the nature of rules of thumb than specific usability guidelines.

- **Visibility of system status**

The system should always keep users informed about what is going on, and what their current status in the system is. In our application this is achieved by several ways.

First of all, this is accomplished with the use of appropriate colors, font size and font weight. For example, the two basic GUI components of the system, Collection Browser and Content Description, are highlighted with the same intense color in order to distract user’s attention. In addition, in the Collection Browser component the selected CHO Collection/CHO is highlighted with gray color and the CHO’s accessibility is shown by italicizing its title text. Finally, the edit mode of a CHO Collection/CHO is shown with light yellow color in the background of the Content Description.

Apart from the color that a CHO Collection/CHO has when it is selected, in order for the user to know which CHO Collection is being displayed, this information is written in the header of the Content Description View. In the case of a CHO Collection selection, the title of the CHO Collection being displayed is written in the header and in the case of a CHO selection its title and the CHO Collection’s title it belongs to are being displayed. Finally, the mode of the CHO Collection/CHO being displayed is also written in the header of the Content Description View so as to inform the user.

- **Match between System and the Real World**

The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms.

In our application, this is achieved with the multilinguality support module that was described in chapter 5. The curators from each museum could have the application in their own language with the user interface terms to be changed according to their preferences. Also, the information box that describes the use of each metadata element in the user interface could be modified and each museum could edit its content by inserting data that match better to the museum exhibits. For example, the metadata element *Creator* for a museum with paintings means the creator of a painting. From the other hand, the same element for a natural history museum has no sense because for instance, there is not a creator of a lizard. *Creator* element for a NHM could be a photographer or a collector. So, this museum could edit the information provided in this metadata element and provide information matching better to its criteria.

- **User Control and Freedom**

Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. That’s why in every user action there are “cancel” options so as to give user the opportunity to leave this state. For example, when a user edits CHO Collection Metadata and selects another CHO Collection from the Collection Browser View by mistake without having saved his changes, the system informs him that he has selected to view another CHO Collection without saving his previous changes. That gives him the option to return to his previous state without losing any data.

- **Consistency and Standards**

The user interface, as was mentioned in the previous section in the basic application structure, consists of a basic pattern that remains such in all the states of the application. All the components have a specific place in the page, in order to not confuse the user with watching the same components in different places in the screen.

- **Error Prevention**

The strategy that was followed for handling the system's errors, during the design of the user interface, was both the prevention and the recover from the errors. This is achieved mainly with the detailed but also short prompts that there are in all form fields, the ability of canceling the user's action and his return in the previous status and finally with the appearance of descriptive error messages.

- **Recognition rather than recall**

A system should minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate. In our application this is accomplished with the use of the information boxes provided for each metadata element in the user interface. The user doesn't need to remember the use of each element or go back to documents to find this information. The system provides an information box for each metadata element which is displaying when the user clicks on the information icon next to the element title. In case of Europeana Semantic Elements, these information boxes contain the official specification as described in the ESE specification [2].

- **Flexibility and efficiency of use**

The use of accelerators may often speed up the interaction for the expert user in such a way that the system can cater to both inexperienced and experienced users. In our application this acceleration is achieved with the use of keyboard shortcuts (like Ctrl-S for Save).

- **Aesthetic and minimalist design**

The user interface of an application should be simple, minimal and not contain information which is irrelevant or rarely needed. The more information that appears on the screen, the less visible each unit of information becomes. In our application the most part of the "screen real estate" is occupied by the Collection Browser widget and the Content Description widget. These widgets contain all the basic information and all the often-executed operations made by the users. Moreover, the metadata elements have been categorized in such a way that the most often used elements are grouped together in the first (selected by default) category and the most rarely used elements in the last category. In addition to this, the metadata elements in each category have been ordered so as to place the most important elements to the top of the list in order to avoid user from scrolling.

Also, the user interface consists of two basic colors (green and gray-silver) in order to be simple and minimal but also pleasant and easy in use. The green color was adopted due to the fact that declares security, accuracy and incitement and exists in some points of the UI because of his intense hue. Gray-silver was adopted because it is a neutral color; it is almost complementary to the green and because of his soft hue. That's the reason why the most elements of the user interface are highlighted with this color. The green color has been used in elements with high importance, as for instance, to indicate the Collection Browser widget or the selected category of the metadata menu.

The buttons of the UI are colored with the green, red and silver color. The silver buttons indicate operations that concern just the user interface. To be more specific, it indicates all the operations that have not any impact to the repository of the application and thus are reversible. Such operations are: the cancellation of user actions, the change of the mode of the CHO/CHO Collection metadata and the change of the media object of the described CHO. The green color as already mentioned declares security, accuracy and incitement. As a result, buttons with these colors prompt for accurate and safe actions. Finally, the red color indicates emphasis and loss. It has been used only in cases of deletion of objects such as the deletion of CHOs or the deletion of a whole CHO Collection.

- **Help users recognize, diagnose and recover from errors**

If a user does an operation and for some reasons an error occurred, the user is always informed via a popup window. The popup window contains descriptive information about what has caused the error and guides the user with specific information in order to recover from the error. To mention that the error messages are not aggressive and there aren't exclamations so as not to intimidate the user and conclude to his demoralization.

- **Help and documentation**

The system supplies a help option in the Main Menu GUI component where the user can find all the information needed in order to complete a specific task. All the possible scenarios have been recorded, with specific screenshots for each scenario provided.

Chapter 7

Implementation

EuroMuse has been implemented as a Web application using the Google Web Toolkit (GWT) platform. A fully functioning instance of the tool has been deployed in the following URL address: http://147.27.41.103:8080/mmat/. The tool provides all the functionality described in the previous sections, and is currently tested by NHM partners. It allows, users of three types (i.e., administrators, curators, and guests) to perform the following actions depending on their access rights:

- Create/delete/manage CHO Collections
- Describe CHO Collections (with appropriate metadata)
- Import/create/delete/manage/export CHO Metadata records
- Import/publish/update/delete CHOs
- Create/delete/update/manage Users

In this chapter, we will describe some implementation issues of the application which are separated in two sections:

- Architecture implementation, based on the System Architecture.
- Graphical User Interface implementation, based on the GUI Specification.

7.1. Architecture

The implementation of the system architecture as described in chapter 5, is comprised of the Client Side and the Server Side logic.

7.1.1. Client Side

The Client Side implementation, which adopts the MVP design pattern, is described thoroughly in the next subsections.

7.1.1.1. Model

The Model of our application consists of the following Java classes that comprise the SIPs and DIPs of the application:

- ***CollectionSIP*****:** Contains all the CHO Collection information that is submitted to the repository.
- ***RecordSIP*****:** Contains all the CHO information that is submitted to the repository.
- ***CollectionDIP*****:** Contains all the CHO Collection information that is disseminated from the repository.
- ***CompactCollectionDIP*****:** Contains a part of the CHO Collection information (id, title, lastModified) that is disseminated from the repository.
- ***RecordDIP*****:** Contains all the CHO information that is disseminated from the repository.
- ***CompactRecordDIP*****:** Contains a part of the CHO information (id, title, status, accessibility) that is disseminated from the repository.
- ***CompactUser***: Contains all the data that characterize a user apart from the password.

7.1.1.2. View

Below, we quote a brief description for each View we have implemented in our application. Each View is a Java class and is connected with its own uiBinder.xml template file in order to define its widgets and its layout.

The Views are:

- ***AdministrationView***: Allows administrators to view and edit all the user profiles and also create a new user account.
- ***ApplicationLayoutView***: Responsible for defining the overall layout of the application.
- ***CollectionBrowserView***: Allows curators to browse collections of CHOs for which they provide descriptions.
- ***CollectionMetadataView***: Allows curators to view, add and edit descriptions (metadata) and update or delete a whole CHO collection.
- ***RecordMetadataView***: Composed of two parts: the part of displaying the metadata elements of a CHO and the part of displaying a preview of this CHO. This view allows curators to add, view and edit descriptions (metadata), to add and edit its multimedia object and update or delete this CHO.
- ***LoginView***: The first view of the application where the users try to login to the system.

- ***MenuBarView***: Allows curators to import record based metadata or ESE metadata to new or existing CHO collection, import media objects (i.e. image, video, sound, text) and export CHO or CHO Collection metadata in ESE format.
- ***UserMenuView***: Allows users to edit their account profile and logout from the system.

7.1.1.3. Presenter

Similarly, we quote a brief description for each Presenter we have implemented in our application and the App events that each presenter generates and is interested to.

- ***AdministrationPresenter***: Sets the user account list to the corresponding View, checks the administrator's input when changing or creating a user account and edits the user accounts.
- ***ApplicationLayoutPresenter***: In control of creating all the other presenters with supplying a View that each Presenter will drive.

Events of Interest

Event Name	Generated By	Data Required	Handling Actions
ShowAdministrationViewEvent	MainMenuPresenter	-	Displays the Administration View.

- ***CollectionBrowserPresenter***: Sets the available CHO Collections to the tree widget, and edits (creates ,deletes, updates) the CHO Collections and CHOs in the tree.

Events Generated

Generated Event	Data Transmitted	Triggering Action
RecordSelectedEvent	collectionID, recordID, title	View CHO description
CollectionSelectedEvent	collectionID, title	View CHO Collection description
GetCollectionEvent(on open)	collectionID	Load CHO Collection

Events of Interest

Event Name	Generated By	Data Required	Handling Actions
ShowAllCollectionsEvent	Main menu widget, Application Manager	List of CompactCollection DIPs	Init collection browser, set the CHO Collection titles in the tree.
RefreshItemEvent	Content description viewer widget	collectionID/ recordID, title	Update a tree item(CHO/CHO Collection)
ShowCollectionEvent	Main menu widget, Application Manager	collectionDIP	Show the CHOS of the CHO Collection

SetRecordToCollectionEvent	Main menu widget	collectionID	Set new CHO to CHO Collection
RecordDeletedEvent	Content description viewer widget, Application Manager	collectionID, recordID	Delete the CHO item from tree.
CollectionDeletedEvent	Content description viewer widget, Application Manager	collectionID	Delete the CHO Collection item from tree and all its CHOs items

- ***ContentMetadataPresenter***: The common Presenter of CollectionMetadataView and RecordMetadataView. Its role is to display the descriptions of the selected CHO Collection/CHO in the metadata element fields, to create, delete and update the displayed CHO Collection/CHO and finally to make all the necessary checks in order to decide if a CHO Collection/CHO is going to be locked or unlocked.

Events Generated

Generated Event	**Data Transmitted**	**Triggering Action**
RecordCreatedEvent	RecordDIP	create new CHO
DeleteCollectionEvent	collectionID, title	delete selected CHO Collection
DeleteRecordEvent	recordID, collectionID, title	delete selected CHO
SaveRecordEvent	recordSIP, recordID, collectionID	save selected CHO
SaveCollectionEvent	collectionSIP, collectionID	save selected CHO Collection
RefreshItemEvent	collectionID, recordID, title	refresh the item in the collection browser widget
RecordSelectedEvent	recordID, collectionID, title	View CHO description
CollectionSelectedEvent	collectionID	View CHO Collection description

Events of Interest

Event Name	**Generated By**	**Data Required**	**Handling Actions**
ShowCollectionMetadataEvent	Application Manager	CollectionDIP	Show the CHO collection metadata
ShowRecordMetadataEvent	Application Manager	recordDIP	Show the CHO metadata
RecordCreatedEvent	Application Manager	RecordDIP boolean	Lock the created CHO

		doNotLock	
RecordDeletedEvent	Collection browser widget, Application Manager	recordID	Unlock deleted CHO. Show the first page.
CollectionDeletedEvent	Collection browser widget, Application Manager	collectionID	Unlock deleted CHO Collection. Show the first page.

- ***LoginPresenter***: Responsible for the validation of the user's input so as to login him into the system.
- ***MainMenuPresenter***: The common presenter of MenuBarView and UserMenuView. In the case of MenuBarView it handles all the operations of importing and exporting metadata, importing media objects and creating CHO Collections or CHOs. In the case of UserMenuView it handles the logout operation and the updating of the logged in user account.

Events Generated

Generated Event	Data Transmitted	Triggering Action
CollectionImportCompletedEvent	collectionID	Show the imported CHO Collection with its CHOS
SetRecordToCollectionEvent	collectionID	Set the new CHO.
ShowAdministrationViewEvent		Show the administration View.
ShowCollectionEvent	CollectionDIP	On create CHO Collection, show the newly created CHO Collection.
EditCollNameToImportListEvent	collectionID, title	Insert the newly created CHO Collection to the existing CHO Collection list.

Events of Interest

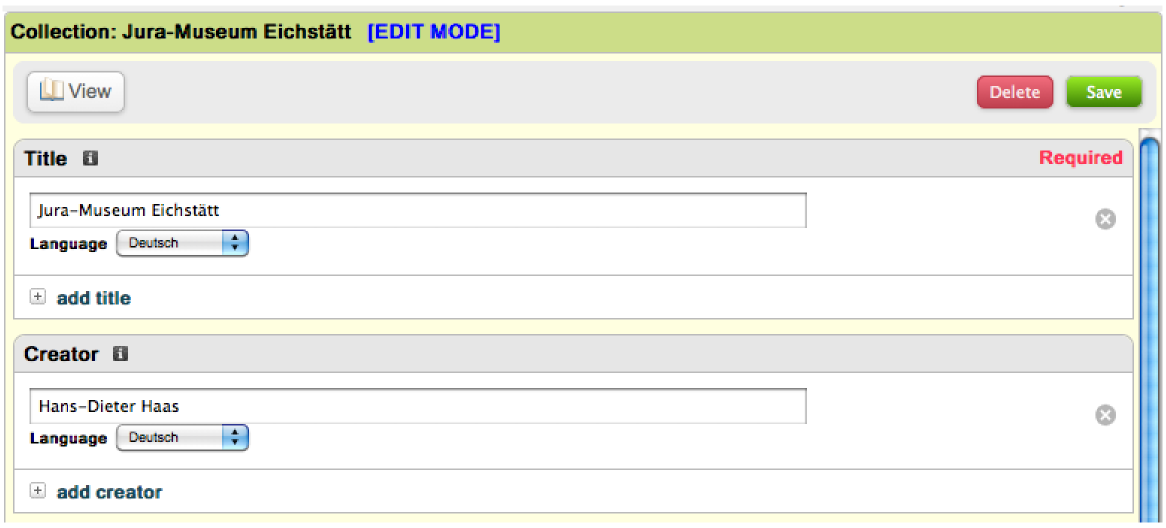
Event Name	Generated By	Data Required	Handling Actions
ShowAllCollectionsEvent	Application Manager	List of CompactCollectionDIP	Initialize the list with the existing CHO Collections.
EditCollNameToImportListEvent	Application Manager, Main Menu	collectionID, title	Edit CHO Collection name to the existing CHO Collection list.

Each of these events extends GwtEvent and for our application we have defined handler interfaces for each of our events.

Binding Presenters and Views

In order to understand better the binding between the Presenters and the Views and how the Display interface works, we will provide a small example of our implementation.

Let's take a part of the CollectionMetadataView:



This part has three buttons and two collection metadata elements. In order for the application to do something meaningful, the presenter is going to need to respond to button clicks, to populate the values of the elements and to get the user entered fields. So, in the case of our ContentMetadataPresenter, we define these five methods in the Display interface as such:

```
HasClickHandlers getSaveButton();
HasClickHandlers getDeleteButton();
HasClickHandlers getModeButton();
void setElementValues(String name, String value);
String getElementValue(String name);
```

The presenter calls these methods in order to handle the button clicks and populate or get the elements values. If user clicks the delete button to delete a collection the presenter handles the user click like this:

```
display.getDeleteButton().addClickHandler(new ClickHandler() {
    public void onClick(ClickEvent event) {
      deleteCollection();
    }
  });
```

Method setElementValues () is a simple way of getting Model data into the View without the View having intrinsic knowledge of the Model itself. The beauty of using setElementValues () is that changes to the model can be made without updating the view code.

Another advantage of such structure is that if we wanted for example to run this application within a mobile browser we could switch out the views without having to change any of the surrounding application code.

7.1.1.4. Multilinguality support

The process we followed for creating the translation for each language supported using the Static String Internationalization technique is straightforward and is described below.

- First, we implemented two Java interfaces for each language supported:
    - one for string constants, the GWT Constants interface (Constants.java)
    - one for parameterized messages, the GWT Messages interface (Messages.java)

  These interfaces use annotations to specify the default translation.

- Then, for each language, we created two Java properties files:
    - one for string constants (Constants_el.properties)
    - one for parameterized messages (Messages_el.properties)

Finally, we replaced all the strings hardcoded in the Java source code with method calls to the appropriate interface.

7.1.2. Server-Side

In this subsection we provide the set of services (CRUD Service, CHO Import Service, Vocabulary Access Service, Concurrency Service) that comprise the middleware concealing the application's business logic from the client. The implemented methods are listed in the tables below, providing brief descriptions regarding each method's functionality.

Method No. 1:	**insertMediaObject**	
Description:	Inserts a media object along with its automatically generated thumbnail images to the db.	
Method Input:		
Parameter Name	*Parameter Type*	*Description*
blob	Byte[]	The media object as blob.
Method Output:	-	

Method No. 2:	insertMediaObject	
Description:	Inserts a media object along with its automatically generated thumbnail images to the db, given the uri pointing the original media object. The original media object is not actually stored in the db.	
Method Input:		
Parameter Name	*Parameter Type*	*Description*
uri	String	The media object's uri.
Method Output:	-	

Method No. 3:	deleteMediaObjectById	
Description:	Deletes a media object along with its thumbnail images (if any) from the db.	
Method Input:		
Parameter Name	*Parameter Type*	*Description*
objectId	String	The media object's identifier.
Method Output:	-	

Method No. 4:	deleteMediaObjectByUri	
Description:	Deletes a media object along with its thumbnail images (if any) from the db.	
Method Input:		
Parameter Name	*Parameter Type*	*Description*
uri	String	The media object's uri.
Method Output:	-	

Method No. 5:	insertRecord	
Description:	Inserts a record SIP to a collection.	
Method Input:		
Parameter Name	*Parameter Type*	*Description*
recordSIP	RecordSIP	A record SIP object.
collectionId	String	The collection’s identifier.
userId	String	The identifier (username) of the user that submitted the record.
Method Output:	-	

Method No. 6:	**updateRecord**
Description:	Updates a record to a collection.
Method Input:	
Method Output:	-

Parameter Name	*Parameter Type*	*Description*
recordId	String	The record's identifier.
recordSIP	RecordSIP	An updated record (SIP form).
collectionId	String	The collection's identifier.
userId	String	The identifier (username) of the user that updated the record.

Method No. 7:	**deleteRecord**
Description:	Deletes a record from a collection.
Method Input:	
Method Output:	-

Parameter Name	*Parameter Type*	*Description*
recordId	String	The record's identifier.
collectionId	String	The collection's identifier.
userId	String	The identifier (username) of the user that deleted the record.

Method No. 8:	**insertCollection**
Description:	Inserts a collection that contains only metadata information and no records to the db.
Method Input:	
Method Output:	-

Parameter Name	*Parameter Type*	*Description*
collectionSIP	CollectionSIP	A collection (SIP form).
userId	String	The identifier (username) of the user that submitted the collection.

Method No. 9:	**insertCollection**
Description:	Inserts a collection having a specific record list to the db.
Method Input:	

Parameter Name	*Parameter Type*	*Description*
collectionSIP	CollectionSIP	A collection (SIP form).
recordSIPList	List<RecordSIP>	The collection’s record list.

userId	String	The identifier (username) of the user that submitted the collection.

Method Output:	-

Method No. 10:	**insertCollection**
Description:	Inserts a collection having a specific record list to the db. The media content described by the collection's records should be specified. For each media object identified in the content, a new record is created and placed to the collection's record list.

Method Input:

Parameter Name	*Parameter Type*	*Description*
collectionSIP	CollectionSIP	A collection (SIP form).
blobName	String	The name of the blob containing the collection’s media content.
blob	Byte[]	The collection's media content as blob.
status	RecordStatusEnum	The status of the records to be created.
access	RecordAccessEnum	The access rights of the records to be created.
userId	String	The identifier (username) of the user that submitted the collection.

Method Output:	-

Method No. 11:	**updateCollection**
Description:	Updates a collection’s metadata information.

Method Input:

Parameter Name	*Parameter Type*	*Description*
collectionId	String	The collection's identifier.
collectionSIP	CollectionSIP	An updated collection (SIP).
userId	String	The identifier (username) of the user that updated the collection.

Method Output:	-

Method No. 12:	**updateCollection**
Description:	Inserts records to a collection's record list.

Method Input:

Parameter Name	*Parameter Type*	*Description*
collectionId	String	The collection's identifier.

recordSIPList	List<RecordSIP>	A list of records to be inserted to the collection's record list.
userId	String	The identifier (username) of the user that updated the collection.

Method Output:	-

Method No. 13:	**updateCollection**
Description:	Inserts records to a collection's record list by specifying the records' media content. For each media object identified in the content, a new record is created and placed to the collection's record list. All the identified media objects are stored in a separate media file collection having the same identifier as the record collection.
Method Input:	

Parameter Name	*Parameter Type*	*Description*
collectionId	String	The collection's identifier.
blobName	String	The name of the blob containing the records' media content.
blob	Byte[]	The records' media content as blob.
status	RecordStatusEnum	The status of the records to be created.
access	RecordAccessEnum	The access rights of the records to be created.
userId	String	The identifier (username) of the user that updated the collection.

Method Output:	-

Method No. 14:	**deleteCollection**
Description:	Deletes a collection from the db.
Method Input:	

Parameter Name	*Parameter Type*	*Description*
collectionId	String	The collection's identifier.
userId	String	The identifier (username) of the user that deleted the collection.

Method Output:	-

Method No. 15:	**insertUser**
Description:	Inserts a user to the db.
Method Input:	

Parameter Name	*Parameter Type*	*Description*

user	User	A user object.

Method Output:	-

Method No. 16:	**updateUser**
Description:	Updates a user to the db.
Method Input:	

Parameter Name	*Parameter Type*	*Description*
user	User	An updated user object.

Method Output:	-

Method No. 17:	**activateUser**
Description:	Activates a user.
Method Input:	

Parameter Name	*Parameter Type*	*Description*
userId	String	The user's identifier.

Method Output:	-

Method No. 18:	**deactivateUser**
Description:	Deactivates a user.
Method Input:	

Parameter Name	*Parameter Type*	*Description*
userId	String	The user's identifier.

Method Output:	-

Method No. 19:	**deleteUser**
Description:	Deletes a user from the db.
Method Input:	

Parameter Name	*Parameter Type*	*Description*
userId	String	The user's identifier.

Method Output:	-

Method No. 20:	**getMediaObjectUri**
Description:	Retrieves the uri of a media object.

Method Input:

Parameter Name	*Parameter Type*	*Description*
objectId	String	The media object's identifier.

Method Output:

Parameter Type	*Description*
String	The requested media object uri.

Method No. 21:	**getSourceThumbnailUri**
Description:	Retrieves the source thumbnail image uri of a media object.

Method Input:

Parameter Name	*Parameter Type*	*Description*
objectId	String	The media object's identifier.

Method Output:

Parameter Type	*Description*
String	The requested source thumbnail image uri.

Method No. 22:	**getThumbnailUri**
Description:	Retrieves the thumbnail uri of a media object.

Method Input:

Parameter Name	*Parameter Type*	*Description*
objectId	String	The media object's identifier.

Method Output:

Parameter Type	*Description*
String	The requested thumbnail uri.

Method No. 23:	**getRecord**
Description:	Retrieves a record from a collection.

Method Input:

Parameter Name	*Parameter Type*	*Description*

recordId	String	The record's identifier.
collectionId	String	The collection's identifier.

Method Output:

Parameter Type	*Description*
RecordDIP	The retrieved record (DIP).

Method No. 24:	**getRecordInESEFormat**
Description:	Retrieves a record in ESE format from a collection.

Method Input:

Parameter Name	*Parameter Type*	*Description*
recordId	String	The record's identifier.
collectionId	String	The collection's identifier.

Method Output:

Parameter Type	*Description*
String	The ESE record.

Method No. 25:	**getCollection**
Description:	Retrieves a collection from the db.

Method Input:

Parameter Name	*Parameter Type*	*Description*
collectionId	String	The collection's identifier.

Method Output:

Parameter Type	*Description*
CollectionDIP	A light version of the collection (DIP form).

Method No. 26:	**getCollection**
Description:	Retrieves a collection from the db. An option for defining the preferred language of the returned record titles is available.

Method Input:

Parameter Name	*Parameter Type*	*Description*
collectionId	String	The collection's identifier.
locale	String	The language option.

Method Output:

Parameter Type	*Description*
CollectionDIP	A light version of the collection (DIP form).

Method No. 27:	**getCollectionInESEFormat**
Description:	Retrieves a collection in ESE format from the db.

Method Input:

Parameter Name	*Parameter Type*	*Description*
collectionId	String	The collection's identifier.

Method Output:

Parameter Type	*Description*
String	The collection’s records in ESE format.

Method No. 28:	**getCompactCollection**
Description:	Retrieves a compact form of a collection from the db. An option for defining the preferred language of the returned collection’s data is available.

Method Input:

Parameter Name	*Parameter Type*	*Description*
collectionId	String	The collection's identifier.
locale	String	The language option.

Method Output:

Parameter Type	*Description*
CompactCollectionDIP	A compact version of the collection (DIP form).

Method No. 29:	**getCollectionList**
Description:	Retrieves a list of the collections (only identifier and title pairs) in the db. An option for defining the preferred language of the returned collection titles is available.

Method Input:

Parameter Name	*Parameter Type*	*Description*
locale	String	The language option.

Method Output:

Parameter Type	*Description*

List<CompactCollectionDIP>	A list of compact collection DIPs.

Method No. 30:	**getUser**
Description:	Retrieves a user from the db.

Method Input:

Parameter Name	*Parameter Type*	*Description*
userId	String	The user's identifier.

Method Output:

Parameter Type	*Description*
User	The retrieved user.

Method No. 31:	**getUserList**
Description:	Retrieves a list of users from the db.
Method Input:	-

Method Output:

Parameter Type	*Description*
List<CompactUser>	A list of users in their compact form.

Method No. 32:	**authenticateUser**
Description:	Authenticates a user.

Method Input:

Parameter Name	*Parameter Type*	*Description*
userId	String	The user's identifier.
password	String	The user’s password.

Method Output:

Parameter Type	*Description*
CompactUser	The compact version of the authenticated user.

Method No. 33:	**lockRecord**
Description:	Acquires lock for a specific record.

Method Input:

Parameter Name	*Parameter Type*	*Description*
collectionId	String	The collection's identifier.
recordId	String	The record’s identifier.
timestamp	long	Timestamp that declares the last modification of the record.

Method Output:

Parameter Type	*Description*
RecordDIP	The record that was locked.

Method No. 34:	**lockCollection**
Description:	Acquires lock for a specific collection.

Method Input:

Parameter Name	*Parameter Type*	*Description*
collectionId	String	The collection's identifier.
timestamp	long	Timestamp that declares the last modification of the collection.

Method Output:

Parameter Type	*Description*
CollectionDIP	The collection that was locked.

Method No. 35:	**unlockResource**
Description:	Releases lock for a specific resource.

Method Input:

Parameter Name	*Parameter Type*	*Description*
id	String	The resource’s identifier.
timestamp	long	Timestamp that declares the last modification of the resource.

Method Output: -

Method No. 36:	**refreshLock**
Description:	Refreshes the lock of a resource.

Method Input:

Parameter Name	*Parameter Type*	*Description*
id	String	The resource’s identifier.

Method Output:	-

Method No. 36:	**suggestClassificationTerm**
Description:	Returns all of the classification terms matching the provided prefix.

Method Input:

Parameter Name	*Parameter Type*	*Description*
prefix	String	The prefix to match.
limit	int	Maximum number of entries to return.

Method Output:

Parameter Type	*Description*
List<NameSuggestion>	A list with all the matching classification terms.

7.2. Graphic User Interface

The following sections describe the steps that a user must follow in order to complete a specific action, by providing GUI screenshots.

7.2.1. Logging in

When the user visits the URL of EuroMuse with any web browser, the login window (Figure 19) is presented, prompting to enter his credentials before continuing to the main screen of the tool.

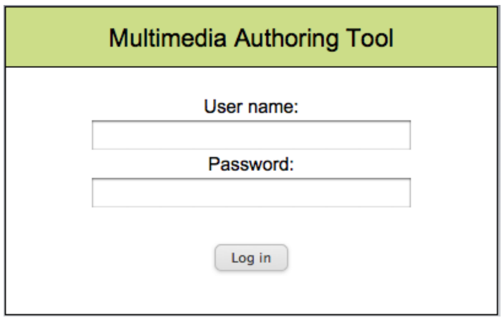


Figure 19: Login window.

7.2.2. Start page

After a successful login, the user is presented with the start page of the tool (Figure 20). The user is able to change the language of the user interface using the pull-down menu on the top-right corner of the page. The homepage of the tool also features a menu bar and a user bar.

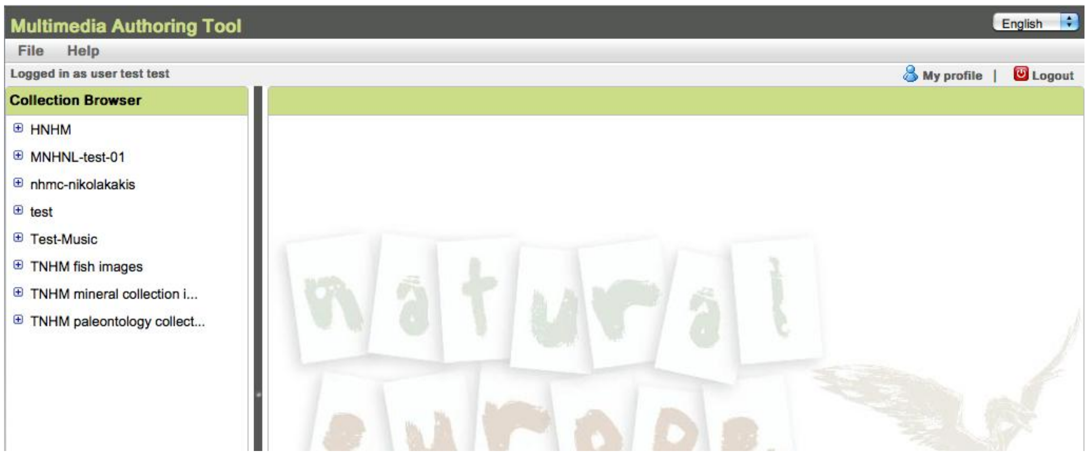


Figure 20: Homepage of the EuroMuse.

On the left of the screen resides the Collection Browser (Figure 21:), containing all the CHO Collections that are already in the system. The user is able to browse the CHO Collections and expand any of them to see the included CHOs.

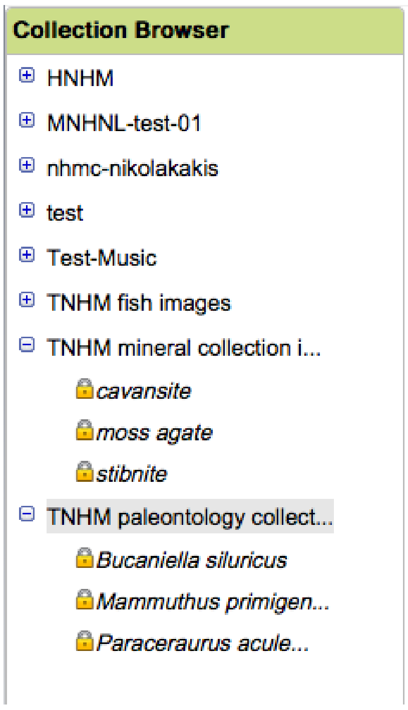
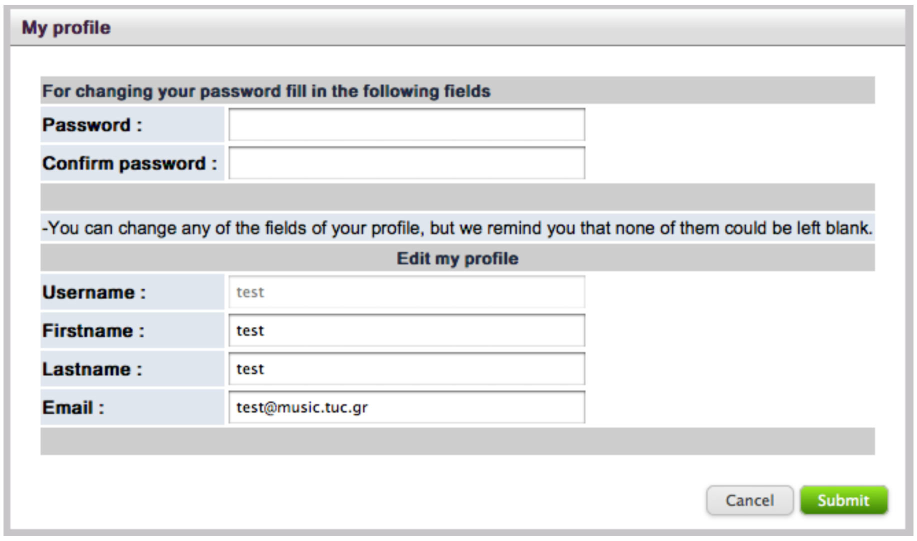


Figure 21: Collection Browser.

7.2.3. Editing user profile

The user is able to edit information related to his profile by clicking the option “My profile” from the user bar on the top right corner of the screen. As a result, a pop up window (Figure 22) prompts the user to change his account details. The permitted user account modifications include (a) changing password by completing the two first fields, or (b) editing personal information (i.e., first name, last name, email).


My profile

For changing your password fill in the following fields

Password :	
Confirm password :	

-You can change any of the fields of your profile, but we remind you that none of them could be left blank.

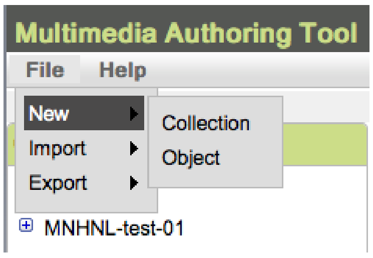
Edit my profile

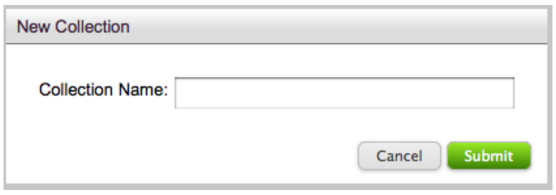
Username :	test
Firstname :	test
Lastname :	test
Email :	test@music.tuc.gr

Figure 22: The User’s Profile View.

7.2.4. Creating/Editing a new CHO Collection

The user is able to create a new CHO Collection by selecting “File -> New -> Collection” from the main menu on the top of the page. As a result, a pop up window prompts the user to provide a name for the new CHO Collection (Figure 23).

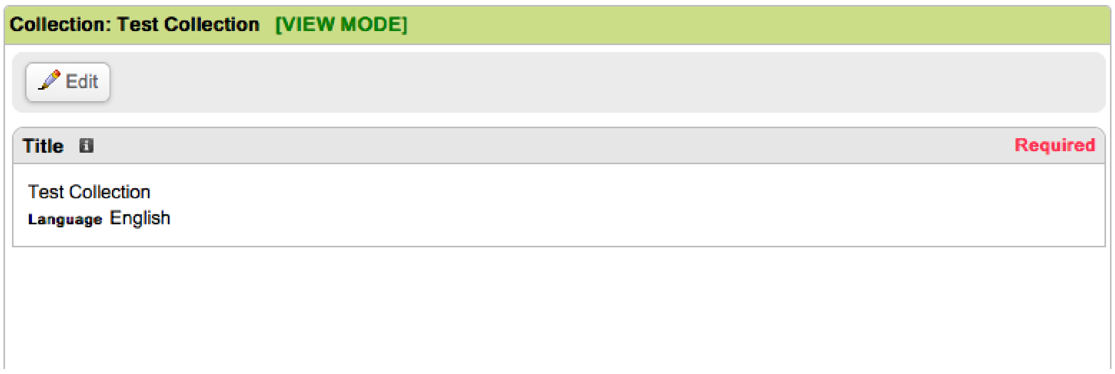



New Collection

Collection Name:

Figure 23: Creating a CHO Collection (left) and providing a name (right).

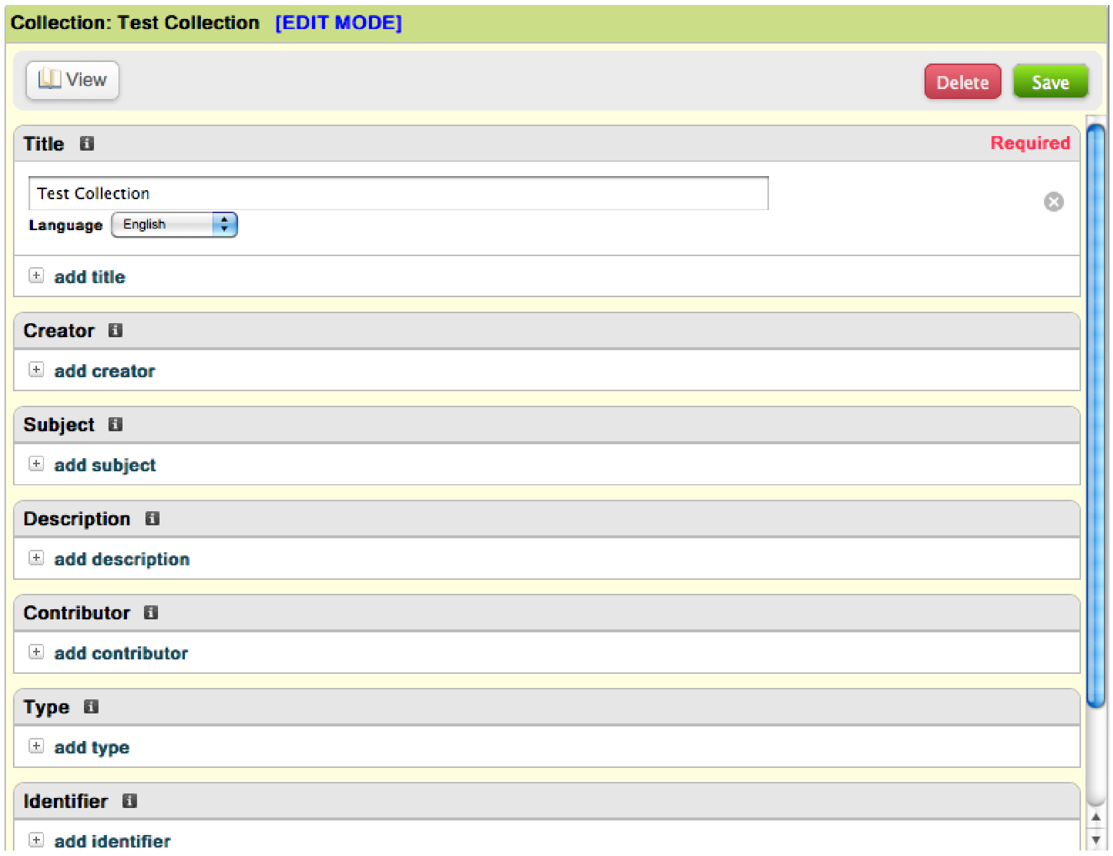
After providing a name, the new CHO Collection is added at the end of the CHO Collection list in the Collection Browser, and the metadata of the CHO Collection (containing only the title for start, as provided by the user) is presented in View Mode (Figure 24). On the top of this page we can see the name of the CHO Collection and the current mode of the Content Description View. Just below resides the CHO Collection’s control bar containing the control buttons and below that are the metadata fields concerning the CHO Collection. The empty fields are hidden from the View Mode, and are displayed only in Edit Mode.



Test Collection
Language English

Figure 24: Metadata information of a CHO Collection in View Mode.

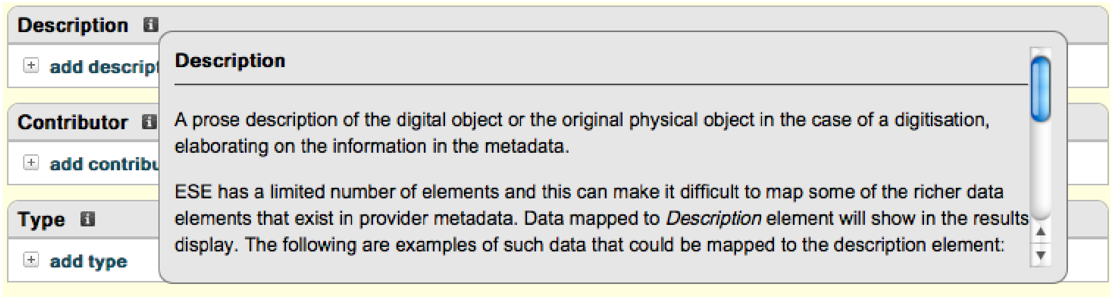
To be able to edit the metadata concerning the CHO Collection, the user needs to click the [Edit] button, which makes the Content Description View enter Edit Mode (Figure 25). The mandatory fields are marked as “required” on the right of the field box.



Test Collection
Language English

Figure 25: Metadata information of a CHO Collection in Edit Mode.

Additional help regarding the information required in each metadata field can be obtained by clicking on the 🛈 icon next to the title (Figure 26).



Description

A prose description of the digital object or the original physical object in the case of a digitisation, elaborating on the information in the metadata.

ESE has a limited number of elements and this can make it difficult to map some of the richer data elements that exist in provider metadata. Data mapped to *Description* element will show in the results display. The following are examples of such data that could be mapped to the description element:

Figure 26: Additional information about the Description metadata field.

When in Edit Mode, the CHO Collection's control bar provides the user with three options (Figure 27):

1. **View:** exits Edit Mode and returns to View Mode. If the user has made any changes, he is prompted to save or discard them before the mode change takes place.
2. **Delete:** deletes the CHO Collection.
3. **Save:** saves the CHO Collection.

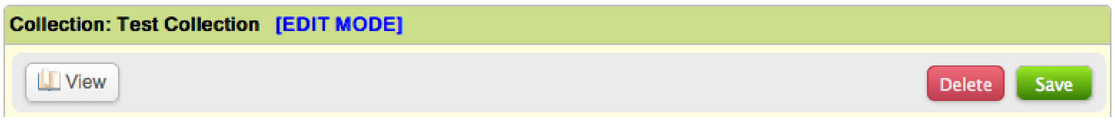


Figure 27: CHO Collection Control bar in Edit Mode.

7.2.5. Creating/editing a new CHO Metadata record

In order to create a new CHO Metadata record, the user must select File -> New -> Object from the menu. The user is then presented with the CHO description in edit mode (Figure 28). The same screen is also displayed when a user changes the mode of an existing CHO Metadata record from view to edit.

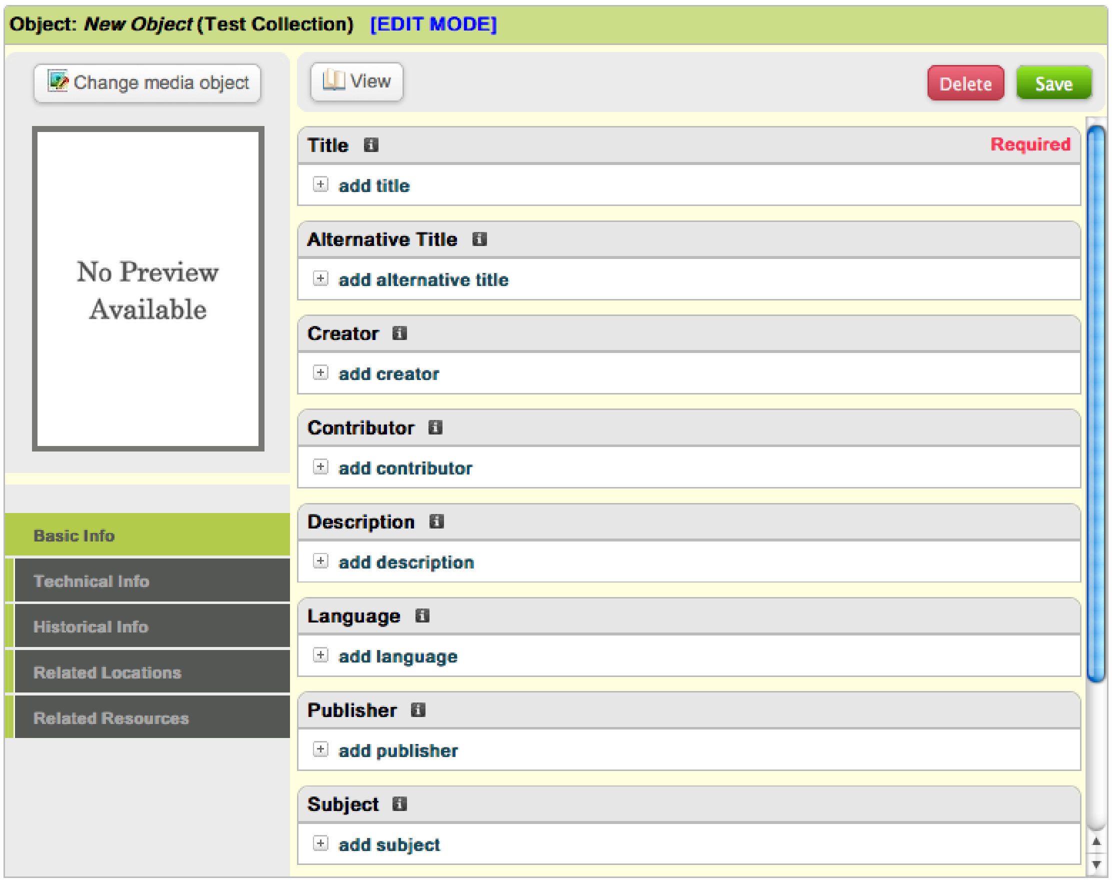


Figure 28: CHO Metadata record in edit mode.

When an object is in Edit Mode, the control bar provides four additional options (Figure 29):

1. **View:** exits Edit Mode and returns to View Mode. If the user has made any changes, he is prompted to save or discard them before the mode change takes place.
2. **Delete:** deletes the CHO Metadata record.
3. **Save**: saves the CHO Metadata record.
4. **Change media object:** a pop up window provides the user with the option to edit the URL of the CHO as well as to upload a CHO stored in the hard disk of his computer (Figure 30).

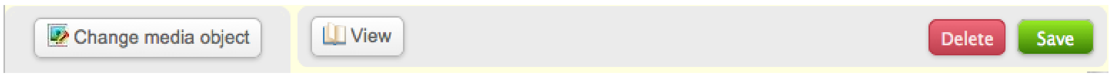
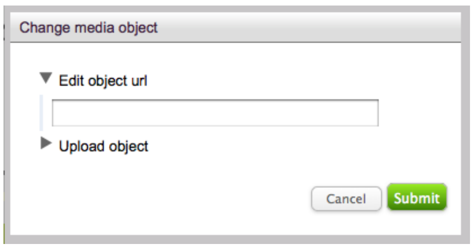


Figure 29: CHO metadata record control bar in Edit Mode.



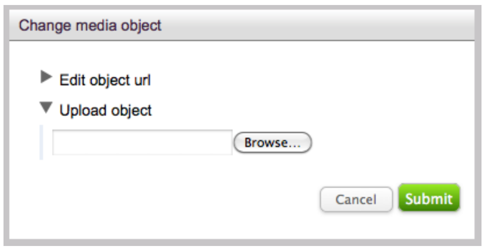


Figure 30: Options for adding an online object (left) and uploading an object (right).

The metadata fields about a CHO are classified in five different categories (Figure 31). When in Edit Mode, the user can enter multiple terms/translations in each metadata field by clicking on the [Add] button at the bottom. An example of providing a title in English and in Greek is shown in Figure 32.

Figure 33, Figure 34, Figure 35, Figure 36, and Figure 37 provide different views for the metadata elements of a fully described CHO, based on the metadata field classification.

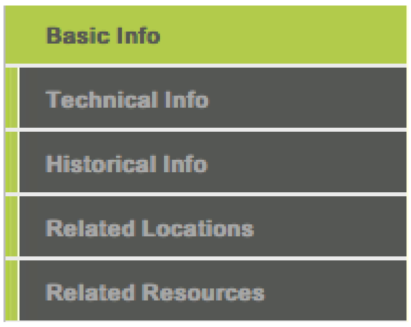
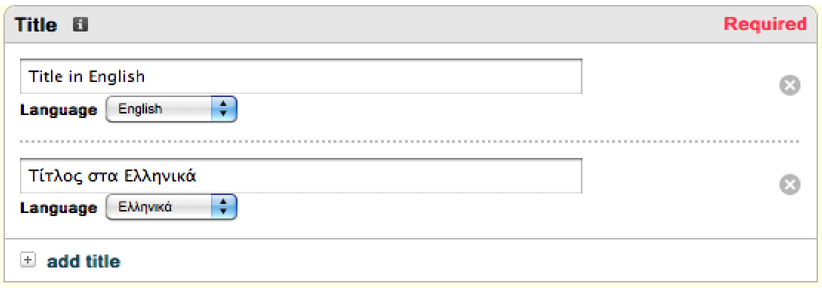


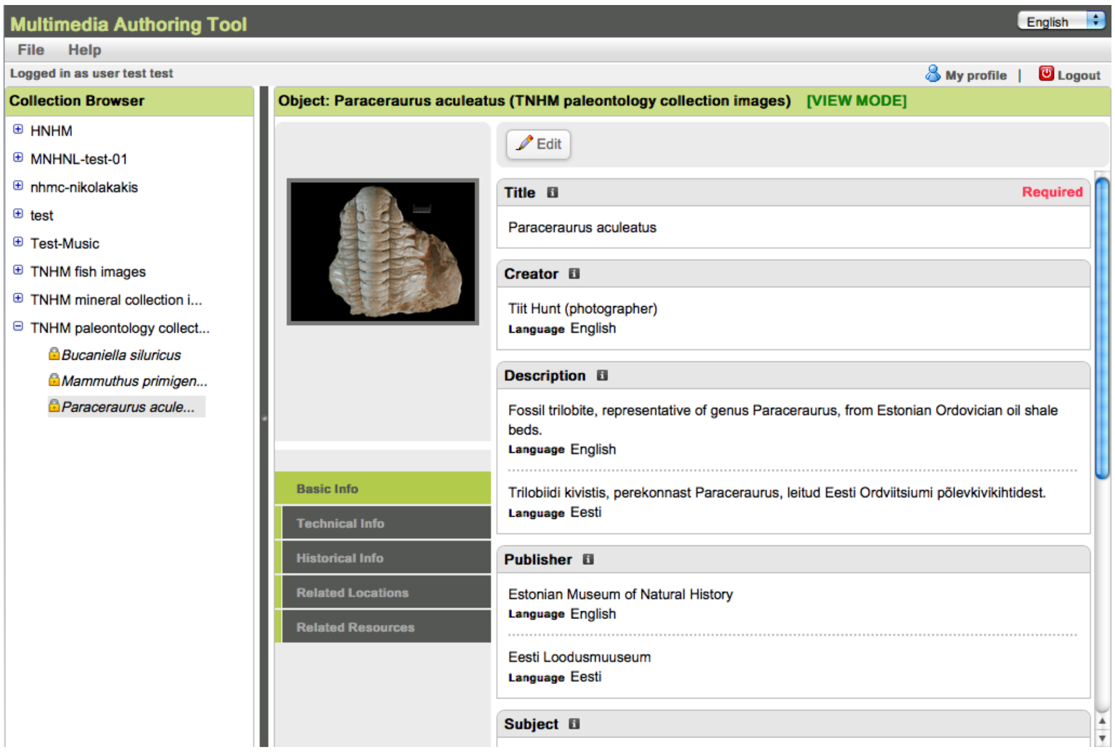
Figure 31: Metadata categories in the CHO Metadata Menu.



Title in English
Language: English

Τίτλος στα Ελληνικά
Language: Ελληνικά

Figure 32: Adding a Greek translation of the English title of the object.



Object: Paraceraurus aculeatus (TNHM paleontology collection images) [VIEW MODE]

Title — Required
Paraceraurus aculeatus

Creator
Tiit Hunt (photographer)
Language English

Description
Fossil trilobite, representative of genus Paraceraurus, from Estonian Ordovician oil shale beds.
Language English

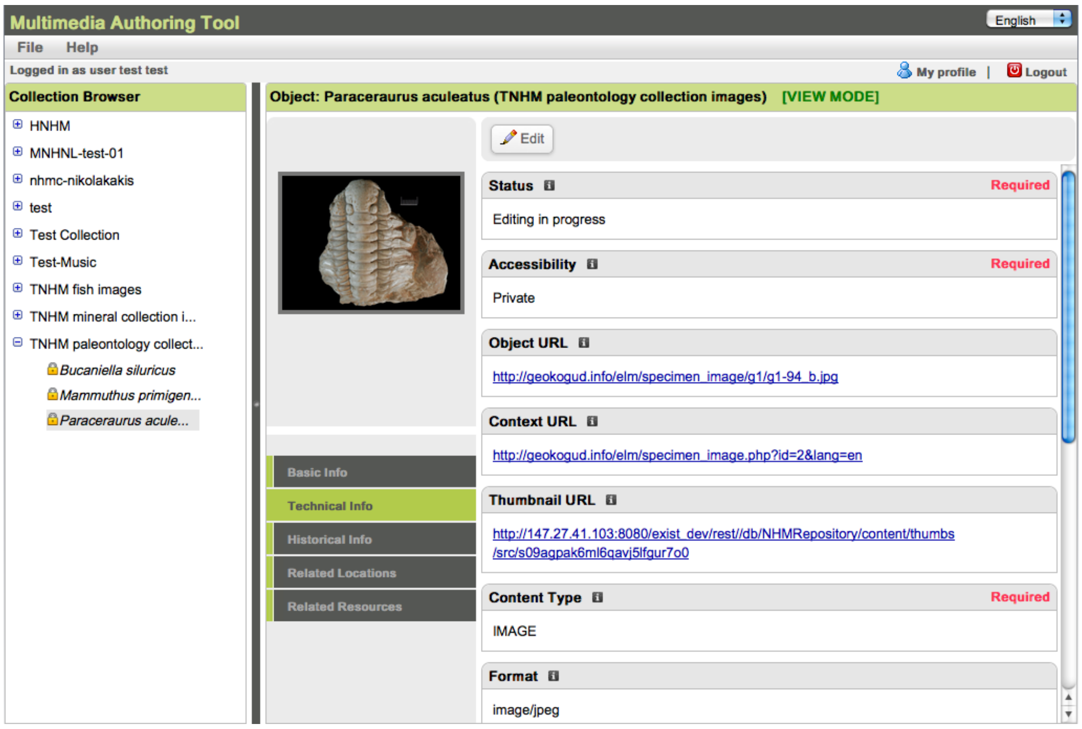
Trilobiidi kivistis, perekonnast Paraceraurus, leitud Eesti Ordviitsiumi põlevkivikihtidest.
Language Eesti

Publisher
Estonian Museum of Natural History
Language English

Eesti Loodusmuuseum
Language Eesti

Subject

Figure 33: Basic Information of a fully described CHO.



Object: Paraceraurus aculeatus (TNHM paleontology collection images) [VIEW MODE]

Status — Required
Editing in progress

Accessibility — Required
Private

Object URL
http://geokogud.info/elm/specimen_image/g1/g1-94_b.jpg

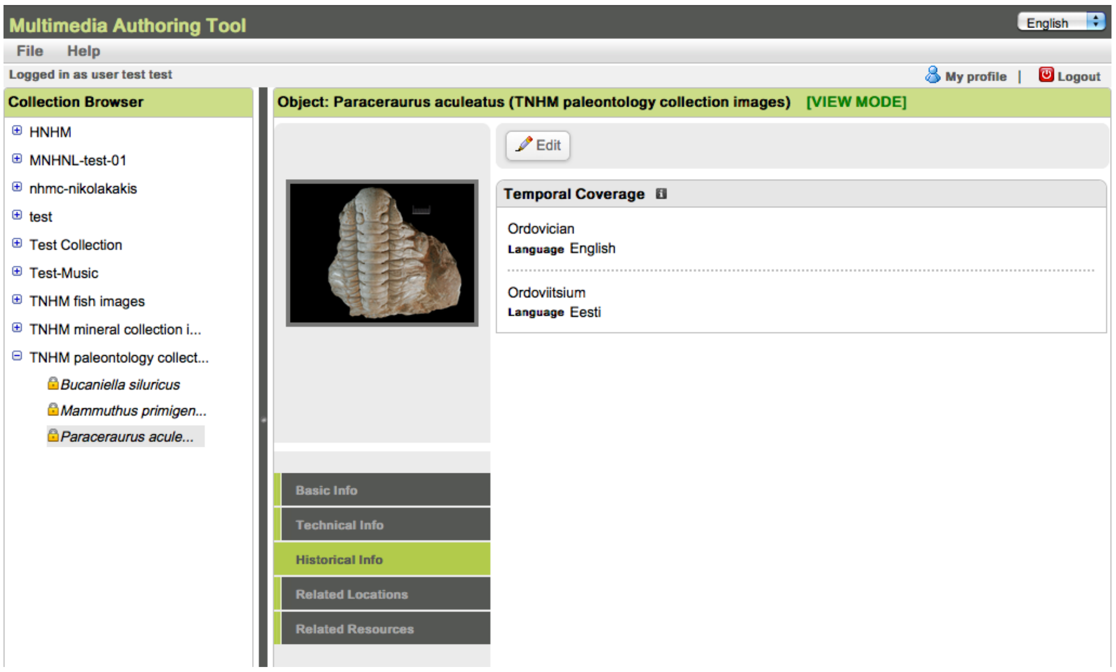
Context URL
http://geokogud.info/elm/specimen_image.php?id=2&lang=en

Thumbnail URL
http://147.27.41.103:8080/exist_dev/rest//db/NHMRepository/content/thumbs/src/s09agpak6ml6gavj5lfgur7o0

Content Type — Required
IMAGE

Format
image/jpeg

Figure 34: Technical information of a fully described CHO.


Multimedia Authoring Tool

English

File Help

Logged in as user test test

My profile | Logout

Collection Browser

- HNHM
- MNHNL-test-01
- nhmc-nikolakakis
- test
- Test Collection
- Test-Music
- TNHM fish images
- TNHM mineral collection i...
- TNHM paleontology collect...
  - *Bucaniella siluricus*
  - *Mammuthus primigen...*
  - *Paraceraurus acule...*

Object: Paraceraurus aculeatus (TNHM paleontology collection images) [VIEW MODE]

Edit

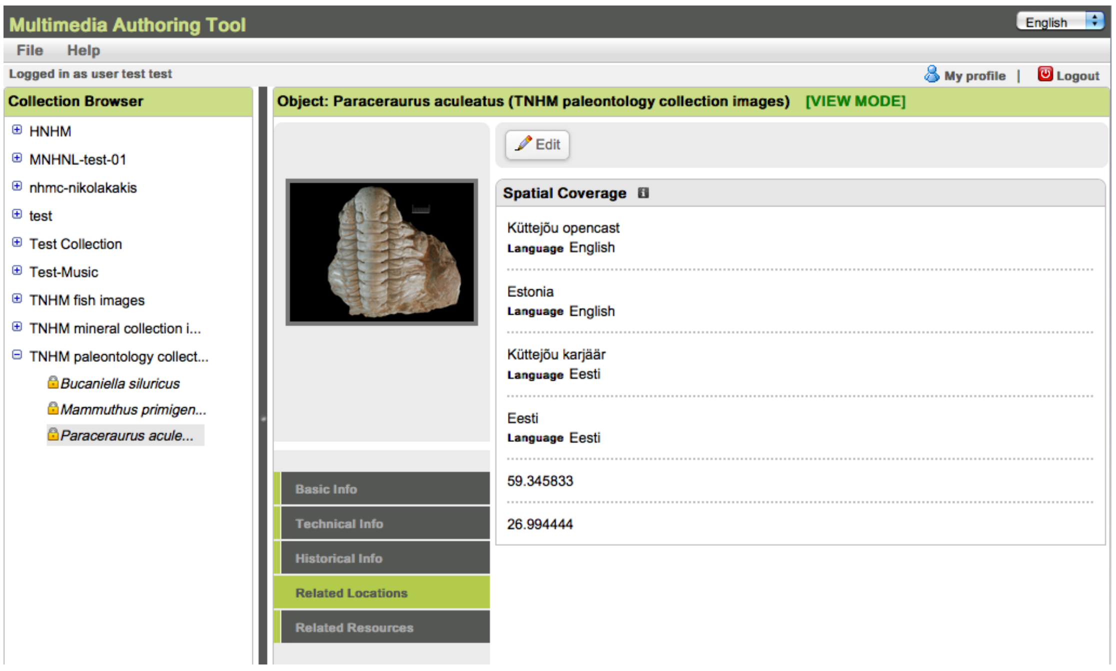
Temporal Coverage

Ordovician
Language English

Ordoviitsium
Language Eesti

Basic Info
Technical Info
Historical Info
Related Locations
Related Resources

Figure 35: Historical Information of a described CHO.


Multimedia Authoring Tool

English

File Help

Logged in as user test test

My profile | Logout

Collection Browser

- HNHM
- MNHNL-test-01
- nhmc-nikolakakis
- test
- Test Collection
- Test-Music
- TNHM fish images
- TNHM mineral collection i...
- TNHM paleontology collect...
  - *Bucaniella siluricus*
  - *Mammuthus primigen...*
  - *Paraceraurus acule...*

Object: Paraceraurus aculeatus (TNHM paleontology collection images) [VIEW MODE]

Edit

Spatial Coverage

Küttejõu opencast
Language English

Estonia
Language English

Küttejõu karjäär
Language Eesti

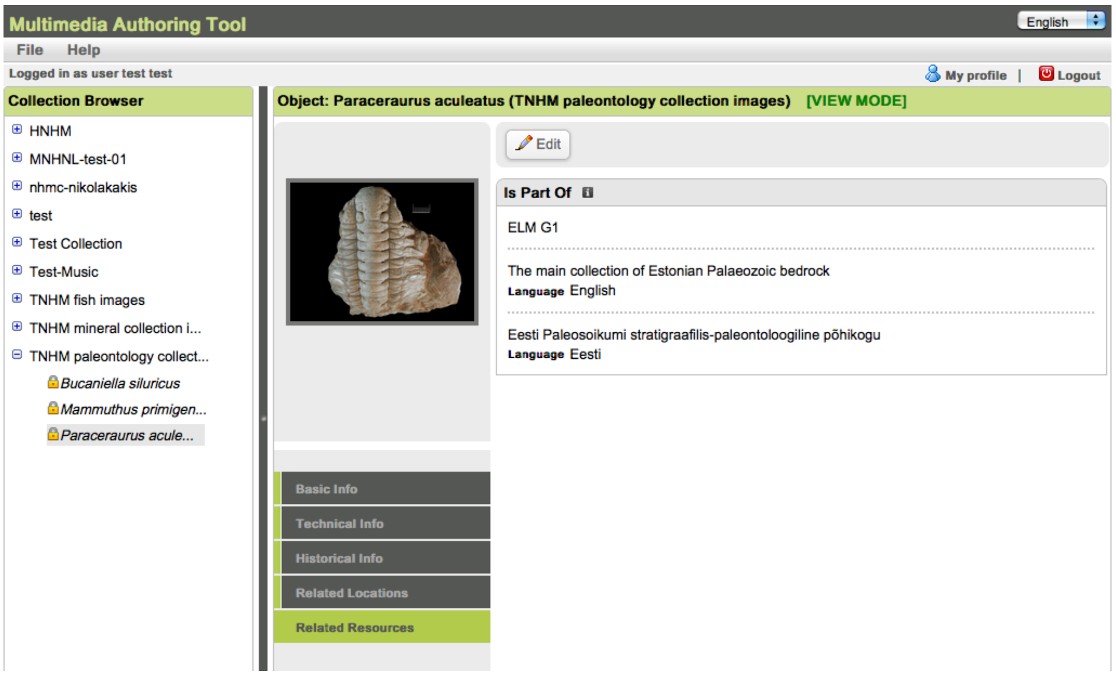
Eesti
Language Eesti

59.345833

26.994444

Basic Info
Technical Info
Historical Info
Related Locations
Related Resources

Figure 36: Related Location Information of a described CHO.


Multimedia Authoring Tool

English

File Help

Logged in as user test test

My profile | Logout

Collection Browser

- HNHM
- MNHNL-test-01
- nhmc-nikolakakis
- test
- Test Collection
- Test-Music
- TNHM fish images
- TNHM mineral collection i...
- TNHM paleontology collect...
  - *Bucaniella siluricus*
  - *Mammuthus primigen...*
  - *Paraceraurus acule...*

Object: Paraceraurus aculeatus (TNHM paleontology collection images) [VIEW MODE]

Edit

Is Part Of

ELM G1

The main collection of Estonian Palaeozoic bedrock
Language English

Eesti Paleosoikumi stratigraafilis-paleontoloogiline põhikogu
Language Eesti

Figure 37: Related Resources Information of a described CHO.

7.2.6. Import options

The user is provided with the option to import Metadata (i.e., CHO Metadata in non ESE format), ESE Metadata (i.e., CHO Metadata in ESE format) or Media Objects (i.e., CHOs), by selecting File -> Import from the main menu (Figure 38).

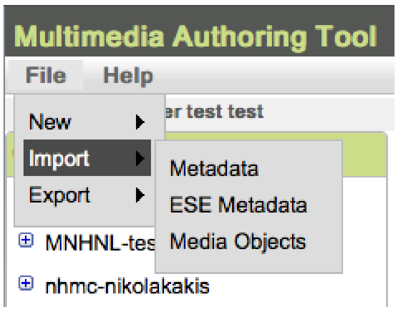


Figure 38: Import options.

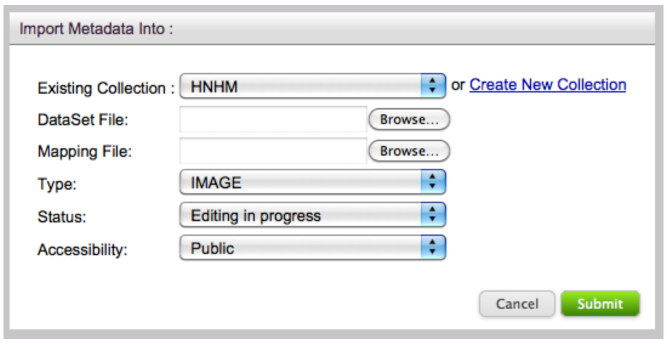
7.2.6.1. Import Metadata

The “Import Metadata” option, allows the user to import metadata in record-based XML format, using a Mapping file created by the Europeana SIP Creator Tool [30]. When selecting “Import Metadata”, the user is presented with the screen shown in Figure 39, prompting to:

1. Select an existing CHO Collection or create a new one.
2. Upload a DataSet & Mapping File from his computer.
3. Define the type of the resource from a pull-down menu (Video, Image, Sound, Text),

4. Describe the status of the process from a pull-down menu (Editing in Process, Editing Complete, Review in Progress, Complete).
5. Define the accessibility level of the resource from a pull-down menu (Public/Private).

When the form is filled correctly and submitted, the system creates the new CHO Collection/Metadata records, which are then added to the Collection Browser so the user can inspect them.


Import Metadata Into :

Existing Collection :	HNHM or Create New Collection
DataSet File:	Browse...
Mapping File:	Browse...
Type:	IMAGE
Status:	Editing in progress
Accessibility:	Public

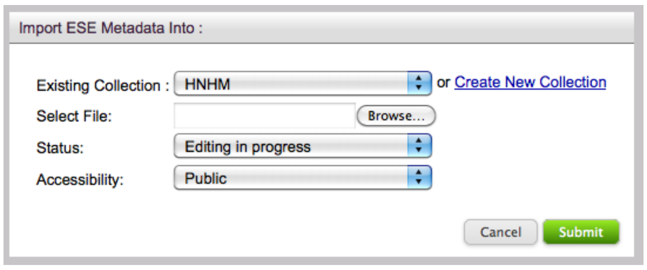
Figure 39: Import metadata options.

7.2.6.2. Import ESE Metadata

The “Import ESE Metadata” option, allows the user to import metadata in ESE format. When selecting “Import ESE Metadata”, the user is presented with the screen shown in Figure 40, prompting to:

1. Select an existing CHO Collection or create a new one in which the imported objects will be inserted.
2. Upload an xml file in ESE format from his computer.
3. Describe the status of the CHO Metadata record being imported from a pull-down menu (Editing in Process, Editing Complete, Review in Progress, Complete).
4. Define the accessibility level of the objects being imported from a pull-down menu (Public/Private).

When the form is filled correctly and submitted, the system creates the new CHO Collection/Metadata records, which are then added to the Collection Browser so the user can inspect them.


Import ESE Metadata Into :

Existing Collection :	HNHM or Create New Collection
Select File:	Browse...
Status:	Editing in progress
Accessibility:	Public

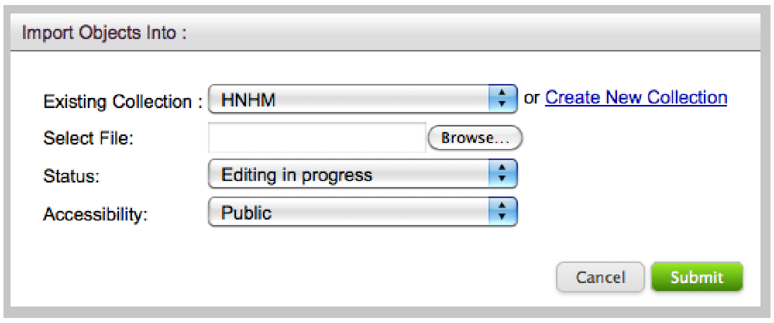
Figure 40: Import ESE metadata options.

7.2.6.3. Import Media Objects

The "Import Media Objects" option, allows the user to import CHOs, which are used from the system to extract the contained metadata (if available). When selecting "Import Media Objects", the user is presented with the screen shown in Figure 41, prompting to:

1. Select an existing CHO Collection or create a new one in which the imported media objects will be inserted.
2. Upload a media object (image/video/sound/text) or a batch of media objects compressed in a .zip file from his computer.
3. Describe the status of the CHOs being imported from a pull-down menu (Editing in Process, Editing Complete, Review in Progress, Complete).
4. Define the accessibility level of the objects being imported (Public/Private).

When the form is filled correctly and submitted, the system creates the new CHO Collection/Metadata records, which are then added to the Collection Browser so the user can inspect them.


Import Objects Into :

Existing Collection :	HNHM or Create New Collection
Select File:	Browse...
Status:	Editing in progress
Accessibility:	Public

Figure 41: Import media object options.

7.2.7. Export options

The user is provided with the option to export the metadata of a CHO Collection's records, or the metadata of a specific CHO in ESE format, by selecting File -> Export -> ESE Metadata (Figure 42). When selecting the "Export ESE Metadata" option, an .xml file containing the ESE Metadata is offered to the user for download.

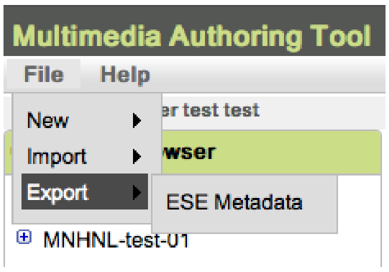


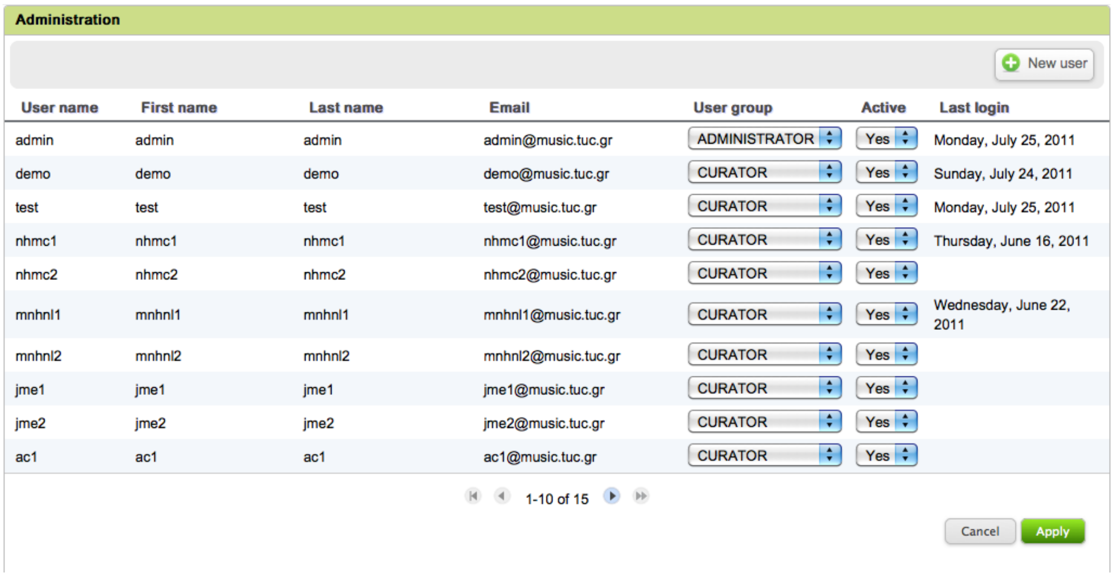
Figure 42: Export options.

7.2.8. Administration

When the user is logged in as administrator, the main menu bar contains an additional “Administration” option (Figure 43).

Figure 43: Menu bar of the tool when logged in as an administrator.

When the user enters the “Administration” menu, he is presented with a page with a list of all the user accounts with their details, and the option to create new user accounts (Figure 44).

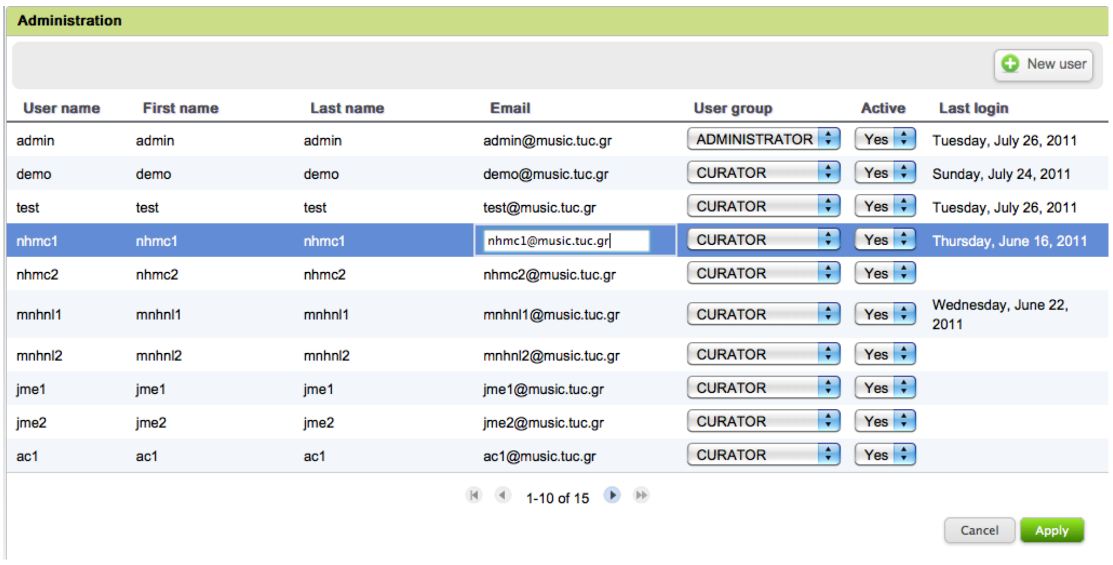


User name	First name	Last name	Email	User group	Active	Last login
admin	admin	admin	admin@music.tuc.gr	ADMINISTRATOR	Yes	Monday, July 25, 2011
demo	demo	demo	demo@music.tuc.gr	CURATOR	Yes	Sunday, July 24, 2011
test	test	test	test@music.tuc.gr	CURATOR	Yes	Monday, July 25, 2011
nhmc1	nhmc1	nhmc1	nhmc1@music.tuc.gr	CURATOR	Yes	Thursday, June 16, 2011
nhmc2	nhmc2	nhmc2	nhmc2@music.tuc.gr	CURATOR	Yes	
mnhnl1	mnhnl1	mnhnl1	mnhnl1@music.tuc.gr	CURATOR	Yes	Wednesday, June 22, 2011
mnhnl2	mnhnl2	mnhnl2	mnhnl2@music.tuc.gr	CURATOR	Yes	
jme1	jme1	jme1	jme1@music.tuc.gr	CURATOR	Yes	
jme2	jme2	jme2	jme2@music.tuc.gr	CURATOR	Yes	
ac1	ac1	ac1	ac1@music.tuc.gr	CURATOR	Yes	

Figure 44: Administration page.

7.2.8.1.Editing user accounts

The administrator is the only person who can edit the user account details except from the password. In order to change one field of someone’s account, he just needs to find the user from the list and click the field he wants to edit (Figure 45). To save the changes made, he has to click the [Apply] button at the bottom right corner of the page.


Administration

New user

User name	First name	Last name	Email	User group	Active	Last login
admin	admin	admin	admin@music.tuc.gr	ADMINISTRATOR	Yes	Tuesday, July 26, 2011
demo	demo	demo	demo@music.tuc.gr	CURATOR	Yes	Sunday, July 24, 2011
test	test	test	test@music.tuc.gr	CURATOR	Yes	Tuesday, July 26, 2011
nhmc1	nhmc1	nhmc1	nhmc1@music.tuc.gr	CURATOR	Yes	Thursday, June 16, 2011
nhmc2	nhmc2	nhmc2	nhmc2@music.tuc.gr	CURATOR	Yes	
mnhnl1	mnhnl1	mnhnl1	mnhnl1@music.tuc.gr	CURATOR	Yes	Wednesday, June 22, 2011
mnhnl2	mnhnl2	mnhnl2	mnhnl2@music.tuc.gr	CURATOR	Yes	
jme1	jme1	jme1	jme1@music.tuc.gr	CURATOR	Yes	
jme2	jme2	jme2	jme2@music.tuc.gr	CURATOR	Yes	
ac1	ac1	ac1	ac1@music.tuc.gr	CURATOR	Yes	

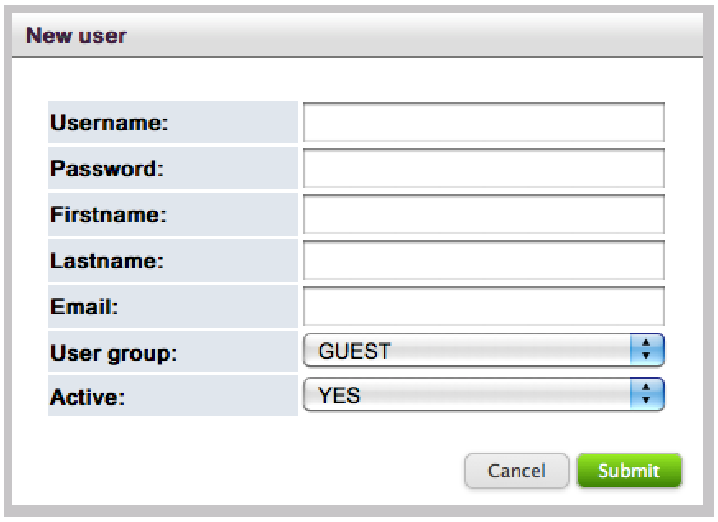
1-10 of 15

Cancel Apply

Figure 45: Editing email field of the user "nhmc1".

7.2.8.2.Creating user accounts

In order to create a new account the administrator has to click on the "New user" button on the top right corner of the administration page (Figure 44) and fill in the form in the pop up window (Figure 46). The administrator needs to enter the user's: (a) username (must be unique), (b) password (can be later changed by the user), (c) first name, (d) last name and (e) email, as well as to define information about the user's (a) group (i.e., administrator, curator, guest) and (b) status (i.e., active or not).


New user

Username:	
Password:	
Firstname:	
Lastname:	
Email:	
User group:	GUEST
Active:	YES

Cancel Submit

Figure 46: Creating a new user account.

Chapter 8

Evaluation

The application was evaluated twice during its development and one time after its first stable release. Firstly, an internal think-aloud evaluation took place in TUC-MUSIC and a hands-on evaluation occurred at Athens in March by the NHMs participating in the Natural Europe project. The problems that emerged from these evaluations have already been overcome. After its first release, it was evaluated during the Natural Europe curators summer school in June in Chania, Crete.

The internal think-aloud evaluation in TUC-MUSIC was performed by giving the laboratory faculty a number of specific tasks to perform, while been observed by us. During the process they were asked to say whatever they were looking for, thinking and feeling while completing these tasks. The problems that emerged from this process were:

- The header of the Content Description widget, in case of a CHO display, contained only the title of the CHO. When a CHO was inside a CHO Collection with many other CHOs, the Collection Browser hadn't enough space to display its corresponding CHO Collection. As a result, the user did not know in which CHO Collection the displayed CHO belongs to. To overcome this issue, in the header of the Content Description widget we added the title of the CHO Collection next to the CHO title.
- The metadata elements were not separated in different categories. As a result, the user had to scroll a lot in order to find a metadata element. To avoid this, we categorize the metadata elements in five categories according to their importance.
- The default language corresponding to the metadata elements was not set. The user of a specific nation due to the fact that the metadata elements he will complete will often be completed to his language, pushed the need of setting as a default language, the language selected for the display of the user interface. If the UI is displayed for instance in Greek, then all the metadata elements will have as a default language the Greek language.

The problems that emerged from the first meeting of the Natural Europe project participants in Athens were:

- The user interface presented flaws in small notebook screens such some scrolling issues.
- In case of a long CHO/CHO Collection title, the user had to scroll a lot in the Collection Browser widget to see all the title. As a result we cut all the titles of more than 25 characters so as to occupy the space of the Collection Browser widget and when the user desires to see all the title he can hover over it.
- When there were a lot of CHOs, there was a difficulty in finding a specific CHO by its title in the Collection Browser widget. This problem has been overcome by placing all the titles in alphabetical order.
- The CHO/CHO Collection title in the Collection Browser widget was taken randomly of all the available titles each expressed in a different language no matter what the selected language of the user interface was. Now, the Collection Browser widget displays the title in the selected language of the UI. If this title doesn’t exist, then it displays randomly one.

In the Natural Europe curators summer school in Chania the only problem that emerged was the appearance of the “OK” button in the popup window when saving the metadata of a CHO. The users characterized it as very annoying due to the fact that it was required to click on the "OK" button every time they pushed the "SAVE" button.

Despite this, the users were very satisfied with the whole application. They characterized it simple and easy to understand. They claimed that the colors of the user interface are very pleasant and not tedious. In addition, they asserted that the application offers them convenience in many issues and proposed some extra features that they want us to support in the future.

Chapter 9

Related Work

In this chapter, we present the most relevant applications that have been implemented to this research field. In particular, we will describe the Collective Access cataloguing tool and the Europeana.eu SIP Creator.

9.1. Collective Access

CollectiveAccess is a highly configurable cataloguing tool and web-based application for museums, archives and digital collections. Available free of charge under the GPL open-source license, it requires little to no custom programming to support a variety of metadata standards, external data sources and repositories, as well as most popular media formats. In addition to multilingual cataloguing facilities, it allows publication of this data in the languages of user's choice. Below are listed the advantages of this application:

- CollectiveAccess is a highly configurable cataloguing tool that doesn't require custom programming. It supports most metadata standards through straightforward configuration. Out-of-the-box support is available for several popular standards including DublinCore, PBCore and SPECTRUM.
- A variety of external data sources and services can be accessed for cataloguing and data display within CollectiveAccess. CollectiveAccess can also integrate with external digital repository systems such as Fedora and IRODS.
- As befits a good world-citizen, CollectiveAccess offers support for multi-lingual cataloguing, as well as translation of the user interface into seven languages (with more on the way). Even better, the CollectiveAccess web-publishing module facilitates creation of multi-lingual web sites. So not only can you catalogue in many languages, you can publish your data in them as well.

- CollectiveAccess can handle a long list of digital media formats, including many popular image, video, audio and document formats. All accepted formats can be automatically re-sized, watermarked and converted to web-viewable formats using criteria the user defines.
- CollectiveAccess is free software released under the GNU Public License, version 2. All of it. There is nothing to buy. There are no "Pro" versions that just happen to cost something. CollectiveAccess is entirely free to use, redistribute, and modify to suit user's needs.
- An incredibly diverse range of collections use CollectiveAccess: from fragments of lost silent films to the latest 4k digital film; from pottery shards to 10m long steel beams; from postcards to performance video. Project-specific setups in the configuration library, our online support forum and user group meetings allow all users to share in the valuable experience gained from these projects.

Although Collective Access supports several popular standards, it doesn't support the Europeana Semantic Elements specification. That means that all the content published to this tool isn't published to Europeana's web portal where increasingly impressive amounts of cultural heritage from various sources are exposed to European citizens. As a result, this content is only accessible from their web-site application.

Also, in case of a museum not having already web-accessible published content, Collective Access doesn't offer the ability to publish it. That means that the museum must find other means to publish its content and make it accessible to the public.

Finally, there are many cases where museums have already existing metadata in some legacy or internal (museum specific) formats that are deprecated or unpublished, and as a result Collective Access doesn't support it. In that case, Collective Access does not give the ability to museum curators to import the already existing metadata and avoid them from writing all the metadata from scratch.

We assume that these are some crucial issues that Collective Access doesn't support due to our experience from contacting the museum curators of the six museums participating in the Natural Europe project.

9.2. Europeana.eu SIP Creator

The SIP-Creator is a standalone Java application designed to enable people to transform any record-based XML input into the format required for ingestion into the Europeana platform. SIP stands for Submission Information Package, and in this context the information submitted will be the original XML file combined with a specification of how it will be mapped to the new format.

The SIP-Creator in order to do the mapping work integrates a general-purpose programming language: **Groovy** language. Groovy is a rich and expressive scripting language which is so related to Java that it is very natural to build it right into a Java application. The SIP-Creator has Groovy integrated in such a way that the user can edit the snippets of Groovy code and see the results of the changes they make immediately. The Groovy code is "live", which makes learning the necessary elements of the language quite easy.

To build a mapping you must go through four stages, indicated by different tabs in the interface.

1. First the source XML file is analyzed, statistics are gathered, and you are asked to select the XML element which separates one record from the next.
2. Then by making selections you build the initial rough mapping, and to make it easier many straightforward mapping choices are made automatically for you.
3. Next, you enter the refinement phase where each mapping can be viewed and edited. This is the only place where you edit Groovy code.
4. Finally the normalization can be done when the mapping is complete, and you can verify that the mapping is correct by stepping through records and seeing the results in a kind of dry-run.

The SIP-Creator has a built-in validation of the resulting record, which reports any problems in the mappings that is likely to produce errors.

EuroMuse offers the same functionality of SIP-Creator in terms of normalizing the source XML files using the mappings provided and producing ESE-CHO application profile compliant documents. The mappings used for the procedure are in the exact same format as SIP-Creator mappings. Although the creation of these mappings is not facilitated by EuroMuse, they can be produced using SIP-Creator, or completely written by hand.

Moreover, EuroMuse allows the seamless importing of the source XML in the web-based system, which instantly offers collaboration support to all museum curators. As SIP-Creator is a standalone application, each curator needs to install it on his computer, whereas using EuroMuse, a curator only needs a browser window, the source XML and the mapping file to perform the normalization.

Chapter 10

Conclusion

In this thesis we presented EuroMuse: a web-based management system, which facilitates the authoring and metadata enrichment of cultural heritage objects in order to be exploited by Europeana. Due to the fact that the management systems of cultural heritage organizations – libraries, museums, archives and audiovisual collections – catalogue their content in different ways and to different standards, EuroMuse supports a rich metadata element set, which includes the Europeana Semantic Elements (ESE), so as to establish the interconnection and interoperability among the museum management systems.

Summarizing, EuroMuse gives the ability to:

- Create, delete, update, and review CHO Collections
- Describe CHO Collections (with appropriate metadata)
- Create, delete, update, review, import and export CHO Metadata
- Import, publish, update, delete CHOs
- Create, delete, update, manage user accounts
- Browse the application in many languages
- Link CHOs with well-established controlled vocabularies

It is developed with the Google Web Toolkit (GWT) framework, in the context of Natural Europe project and is actively being used by the cultural museum experts of six European Natural History Museums who have evaluated the system and have already described over 1000 fully described cultural heritage objects.

Works Cited

1. Europeana. http://europeana.eu/portal/.

2. Europeana Semantic Elements specifications v3.4. http://version1.europeana.eu/c/document_library/get_file?uuid=77376831-67cf-4cff-a7a2-7718388eec1d&groupId=10128.

3. Dublin Core. http://dublincore.org/documents/.

4. **Walmsley, David C. Fallside and Priscilla.** XML Schema. *W3C Recommendation, 2004.* http://www.w3.org/TR/xmlschema-0/.

5. *Ajax: A New Approach to Web Applications.* **Garrett, Jesse James.** 2005.

6. **Fielding, Roy T., et al.** RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1. June 1999. http://tools.ietf.org/html/rfc2616.

7. **Dave Raggett, Arnaud Le Hors, Ian Jacobs.** HTML 4.01. *W3C Recommendation, 24 December 1999.* http://www.w3.org/TR/html4/.

8. Tomcat. http://tomcat.apache.org/.

9. Extensible Markup Language (XML) 1.0 (Fifth Edition). *W3C Recommendation 26 November 2008.* http://www.w3.org/TR/2008/REC-xml-20081126/.

10. W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. *W3C Candidate Recommendation 21 July 2011.* http://www.w3.org/TR/xmlschema11-1/.

11. XMLBeans. http://xmlbeans.apache.org/.

12. Lucene. http://lucene.apache.org/java.

13. Solr. http://lucene.apache.org/solr/.

14. **Frank Manola, Eric Miller.** RDF. *W3C Recommendation, 10 February 2004.* http://www.w3.org/TR/rdf-primer/.

15. **Alistair Miles, Sean Bechhofer.** SKOS. *W3C Recommendation, 18 August 2009.* http://www.w3.org/TR/skos-primer.

16. **Peter F. Patel-Schneider, Patrick Hayes and Ian Horrocks.** OWL Web Ontology Language Semantics and Abstract Syntax. *W3C Recommendation, 10 February 2004.* http://www.w3.org/TR/owl-semantics/.

17. **Carroll, Graham Klyne and Jeremy J.** Resource Description Framework (RDF): Concepts and Abstract Syntax. *W3C Recommendation, 10 February 2004.* http://www.w3.org/TR/rdf-concepts/.

18. **Berners-Lee, David Beckett and Tim.** Turtle - Terse RDF Triple Language. *W3C Team Submission, 14 January 2008.* http://www.w3.org/TeamSubmission/turtle/.

19. Google Web Toolkit (GWT). http://code.google.com/intl/el-GR/webtoolkit/.

20. Biodiversity Heritage Library (BHL). http://www.biodiversityheritagelibrary.org/.

21. Organic.Edunet Portal. Learning Material on Organic Agriculture in Europe. http://portal.organic-edunet.eu/.

22. Europeana Data Model Definition V.5.2. http://version1.europeana.eu/c/document_library/get_file?uuid=aff89c92-b6ff-4373-a279-fc47b9af3af2&groupId=10605.

23. SIP-Creator. http://europeanalabs.eu/sip-creator/.

24. ImageMagick . http://www.imagemagick.org/script/index.php.

25. FFmpeg. http://www.ffmpeg.org/.

26. The Open Archives Initiative Protocol for Metadata Harvesting. http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm.

27. **Potel, Mike.** MVP: Model-View-Presenter. The Taligent Programming Model for C++ and Java. 1996.

28. **Reenskaug, Trygve.** MVC XEROX PARC 1978-79.

29. ISO 14721:2003 Open Archival Information System (OAIS). http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=24683 .

30. Europeana Labs: SIP Creator. http://europeanalabs.eu/sip-creator/.

31. Catalogue of Life. http://www.catalogueoflife.org/.

32. D2R Server. http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/.

33. exist-db . http://exist.sourceforge.net/.

34. http://en.wikipedia.org/wiki/Optimistic_concurrency_control.

35. Europeana Semantic Elements Specification V.3.3. http://version1.europeana.eu/c/document_library/get_file?uuid=a830cb84-9e71-41d6-9ca3-cc36415d16f8&groupId=10602.

36. Europeana Semantic Elements Specifications V3.3. http://version1.europeana.eu/c/document_library/get_file?uuid=a830cb84-9e71-41d6-9ca3-cc36415d16f8&groupId=10602.

37. Java. http://www.oracle.com/technetwork/java/index.html.

38. GWT. http://code.google.com/webtoolkit/.

39. JSON (JavaScript Object Notation). http://www.json.org/.

40. Creative Commons. http://creativecommons.org/licenses/.

Appendix

The ESE-CHO AP Full Element Set

The Natural Europe ESE-CHO AP Full Element Set is a hierarchy of data elements, including container data elements and simple data elements. For each data element, the ESE-CHO AP Schema defines:

- *Default Logical Name:* the name by which the data element is appeared in the Multimedia Authoring Tool;
- *Element Name*: the name by which the data element is referenced;
- *Description*: a short description of the use of the data element;
- *Multiplicity*: the number of values allowed; the size is given as a single number - e.g. '1' - or as a range – e.g. '0..*' - where '*' indicates an unbounded maximum.
- *Value space*: the set of allowed values for the data element – typically in the form of a vocabulary or a reference to another standard;
- *Datatype*: indicates whether the values are LangString, DateTime, VocabularyTerm, or String.
- *Obligation*: indicates whether an element is mandatory or recommended. All elements that do not apply to one of these two categories are considered optional. In some cases obligation describes the mapping of an element to a metadata standard. Also, it can be used to explain the relations between different elements (e.g., if an element is used, then another one becomes mandatory, etc.).
- *Example*: an illustrative example of the element.
- *Method of input*: describes the method (manual or automatic) of input for the elements.

The ESE-CHO APPLICATION PROFILE

Default Logical Name	*Element*	*Description*	*Multiplicity*	*Value Space*	*Data Type*	*Obligation*	*Example*	*Method of input*
Basic Info	**1. Basic**	Groups the general information that describes this CHO object as a whole.						
Title	***1.1 title***	The title of the original analog or born digital object.	1..*		LangString	**Mandatory data element.**	Cyprinus carpio (Language Latin) Wolf drinking water in natural habitat (Language English)	**Automatic** if chosen from element 1.11 classification, then manually (User always has the option to override)
Alternative Title	***1.2 alternative***	Any alternative title by which the original analog or born digital object is known. This can include abbreviations or translations of the title, including	0..*		LangString	**Recommended data element.**	carp (Language English) Karpkala (Language Eesti)	**Automatic** if chosen from element 1.11 classification, then manually (User always has the option to override)

		common names (in the case that the title is the 1.11 classification latin term)						
Creator	***1.3 creator***	This is the name of the creator of the original analog or born digital object. NOTE: This element should not be confused with the element 1.4 contributor.	0..*	use a consistent form of the name e.g. Shakespeare, William.	LangString	**Recommended data element.**	In the case of an animal photograph one could consider as Creator the photographer and as Contributor the person that has processed this photograph. In the case of an exhibit photograph one could consider as Creator the person that set up this exhibit and as Contributor the photographer of this exhibit.	Manually

Contributor	***1.4 contributor***	The name of contributors to the original analog or born digital object. This could be a person, an organisation or a service.	0..*	use a consistent form of the name e.g. Shakespeare, William.	LangString	**Recommended data element.**	Friedrich Schmidt (collector) Tiit Hunt (photographer)	Manually
	1.4.1 role	Kind of contribution.	1	**Please consult the MARC Code List for Relators, (available at id.loc.gov/vocabulary/relators.html) for possible roles.**	VocabularyTerm	**Mandatory data element IF 1.4 contributor is used. Minimally, the Contributor(s) of the CHO object should be described.**		Default "Photographer" then manually
Description	***1.5 description***	A description of the original analog or born digital object.	1..*		LangString	**Mandatory data element.**	Crystal aggregate of mineral cavansite. Cavansite occurs as secondary mineralization product in basalts and	Manually

							andesites. (Language English) Mineraal kavansiit esineb sekundaarse mineraalina Basaltides ja andesiitides. (Language Eesti)	
Language	***1.6 language***	The language of text included in images. If there is no language aspect to the digital object (e.g. a photograph), please ignore this element. NOTE: This element is not for the language of the metadata of a resource.	0..*	Use a 3 letter code from ISO 639-2 (provided)	string	**Strongly Recommended data element where available. Mandatory data element if 1.9 is "Text"**	est	Manually
Publisher	***1.7 publisher***	The name of the publisher (the entity responsible for making the resource available) of the	0..*	In case of a person use a consistent form of the name e.g	LangString	**Recommended data element.**	Estonian Museum of Natural History (Language English)	Manually

		original analog or born digital object.		Shakespeare, William.			Eesti Loodusmuuseu m (Language Eesti)	
Subject	***1.8 subject***	This is the subject of the original analog or born digital object	1..*		LangString	**Mandatory data element.**	Animalia Chordata Actinopterygii cypriniformes cyprinidae Cyprinus	**Automatic** if chosen from element 1.11 classification, then manually (User always has the option to override)
Type	***1.9 type***	The type of the original analog or born digital object as recorded by the content holder. Type includes terms describing general categories, functions, genres, or aggregation levels for content.	0..*	Preferably the values should be taken from the controlled DC Type vocabulary (provided Vocabulary)	LangString or VocabularyTer m	**Recommended data element.**	Image Or Living organism (Language English) Elusorganism (Language Eesti)	Manually
Provenance	***1.10 provenance***	This relates to the ownership and custody of the original analog or	0..*		LangString	**Optional data element**	Donated by the National Library In 1965 (Language	Manually

		born digital object.					English)	
Classification	***1.11 classification***	This relates to the latin term of the original or born digital object.	0..1	SKOS concept(s) from the Natural Europe SKOSified classification scheme	VocabularyTerm	**Strongly recommended data element**	Cyprinus carpio (Language Latin)	Manually
	2. Life Cycle	Describes the history and current state of this CHO object and those entities that have affected this object during its evolution using the Multimedia Authoring Tool (as well as the status of this CHO object within the underlying repository).						
	2.1 version	The edition of this object.	1		string	**Mandatory data element.**	1.2.alpha	Manually

Status	***2.2 status***	The completion status or condition of this object	1	Editing in progress Editing complete Review in progress Complete	VocabularyTerm	**Mandatory data element.**		Manually
	2.3 log	Those entities (i.e., person, organization, service) that have contributed to the state of this object during its life cycle (e.g., creation, editing, publication).	0..*		string	**System generated**	Tiit Hunt (Content Publishing), “2003-03-13”	
	2.3.1 action	Kind of contribution.	1	CHO Creation Content Publishing CHO Annotation CHO Validation	VocabularyTerm			**Automatic,** based on the user action.
	2.3.2 actor	The identification of and information about entities contributing to this object.	1..*	use a consistent form of the name e.g. Shakespeare,	string			**Automatic,** based on the user account information

				William.				
	2.3.3 date	The date of the contribution.	1		DateTime			**Automatic,** based on the system's date
Technical Info	**3. Technical**	This category describes the technical requirements and characteristics of this CHO object.						
Object URL	***4.1 uri***	An unambiguous URL reference to the digital object on the provider's web site in the best available resolution/quality. This is a URL that will be active in the Europeana interface. It will lead users to the digital object on the provider's website where they can view or play it. The digital object needs to be directly accessible by the URL	0..1	valid URL	URI	**Mandatory data element** **Either 4.1 or 4.2 is mandatory.**		**Automatic** whenever possible. For harvested material the information is automatically imported.

		and reasonably independent at that location. Use 4.2 for digital objects embedded in HTML pages (even where the page is extremely simple).						
Context URL	***4.2 contextUri***	An unambiguous URL reference to the digital object on the provider's web site in its full information context. This is a URL that will be active in the Europeana interface. It will lead users to the digital object displayed on the provider's web site in its full information context. Use 4.2 if you display the digital object with extra information (such as header, banner etc).	0..1	valid URL	URI	**Mandatory data element** **Either 4.1 or 4.2 is mandatory.**		**Automatic** whenever possible. For harvested material the information is automatically imported.

Thumbnail URL	***4.3 sourceThumbnailUri***	The URL of a thumbnail representing the digital object or, if there is no such thumbnail, the URL of the digital object in the best resolution available on the web site of the data provider from which a thumbnail could be generated.	1	valid URL	URI	**Mandatory data element**		**Automatic** whenever possible. For harvested material the information is automatically imported.
Content Type	***4.4 resourceType***	The Europeana material type of the resource	1	TEXT IMAGE SOUND VIDEO	VocabularyTerm	**Mandatory data element**		**Automatic** whenever possible. User always has the option to override. For harvested material the information is simply imported or if not given tried to be automatically detected.

Format	***4.5 format***	This unqualified element includes file format, physical medium or dimensions of the original and/or digital object. Use of the more specific elements 4.6 extent (dimensions) and 4.7 medium (physical medium) is preferred where appropriate.	0..*	Internet media types (originally called MIME types) based on IANA registration (provided)	string	**Optional data element**	image/jpeg	**Automatic** whenever possible. User always has the option to override. For harvested material the information is simply imported or if not given tried to be automatically detected.
Extent	***4.6 extent***	The size or duration of the digital object and the original object.	0..*		LangString	**Optional data element**	1280 x 827 pixels 42.4 cm x 68 cm (Language English)	**Automatic** whenever possible. User always has the option to override. For harvested material the information is simply imported or if not given tried to be automatically detected.
Medium	***4.7 medium***	This is the medium of the original analog or	0..*		LangString	**Optional data element**	metal (Language	Manually

		born digital object.					English)	
identifier	***4.8 identifier***	This is the identifier for the original analog or born digital object	0..*		string	**Optional data element**	http://geokogud.info/elm/specimen_image/g297/g297-13_b.jpg	Manually
Rights	***4.9 rights***	Information about intellectual Property Rights, access rights or license arrangements for the digital object (digitized or born digital). The value in this element can be any additional information about intellectual property rights, access rights or license arrangements for the digital object that has not been captured in the controlled value in 8.4 licenceUri element.	1..*		LangString	**Optional data element**	Copyright © Natural History Museum of Crete (Language English)	Manually

Table Of Contents	***4.10 tableOfContents***	A list of the units within the original analog or born digital resource object.	0..*		string	**Optional data element**	Chapter 1. Introduction, Chapter 2. History	Manually
Historical Info	**5. Historical**	This category describes the spatial and temporal characteristics of this CHO object.						
Date	***5.1 date***	A point or period of time associated with an event in the lifecycle of the object. Use for a significant date in the life of the original analog or born digital object. Use element 5.4 Temporal Coverage if the date is associated with the topic of the resource.	0..*		LangString	**Mandatory data element. Use of ISO 8601 starting with the year and hyphenating the day and month parts: YYYY-MM-DD. Either 5.1 or 5.2 or 5.3 is mandatory.**	“1933-12-24”	Manually
Date Created	***5.2 created***	This is the date of the creation of the digital object or, in the case of a digitisation, the	0..*		LangString	**Mandatory data element. Either 5.1 or 5.2 or 5.3 is**	“1935”	Manually

		original physical object. A refinement of 5.1.				**mandatory.**		
Date Issued	***5.3 issued***	The date when the digital object was formally issued or published. This is likely to be the date the original physical object was issued in the case of a digitisation. A refinement of 5.1.	0..*		LangString	**Mandatory data element. Either 5.1 or 5.2 or 5.3 is mandatory.**	"2011-07-07"	Manually
Coverage	***5.5 coverage***	Coverage can be used for either spatial for temporal aspects of the object being described. Values will typically include either a spatial location (place name or geographic co-ordinates) or a temporal period (a date range or period label). If analysis of the data shows that	0..*		LangString	**Recommended data element.**	Boston, MA (Language English)	Manually

		it contains only spatial or only temporal data then please map to either 5.4 or 5.6 elements.						
Temporal Coverage	***5.4 temporal***	The temporal characteristics of the original analog or born digital object i.e. what the resource is about or depicts in terms of time. This may be a period, date or date range. This is in contrast to 5.1 date which relates to an event in the life of the object itself (e.g. the creation or the art work or publication of the book.)	0..*		LangString	**Recommended data element.**	Pleistocene – Holocene (Language English)	Manually

Spatial Coverage	***5.6 spatial***	Information about the spatial characteristics of the original analog or born digital object, i.e. what the resource represents or depicts in terms of space. This may be a named place, a location, a spatial coordinate or a named administrative entity.	1..*		LangString	**Mandatory data element.**	Aardla (Language English) Estonia (Language English) Aardla (Language Eesti) Eesti (Language Eesti) 59.410556, 24.308889	Manually
Related Resources	**6. Relation**	This category defines the relationship between this CHO object and other CHO objects, if any.						
Is Part Of	***6.1 isPartOf***	Use for the name of the collection which the digital object is part of (physically or logically).	0..*		LangString	**Recommended data element.**	Collection of fish images by Tiit Hunt (Language English) Tiit Hunt'i kalafotod (Language	Manually

							Eesti)	
Source	***6.2 source***	This element can be used for several different types of sources that are related to the object (such as reference sources)	0..*		LangString	**Recommended data element.**	BAM portal (Language English) Security Magazine pp 3-12 (Language English)	Manually
Relation	***6.3 relation***	This is information about resources that are related to the original analog or born digital object.	0..*		string	**Optional data element**	maps.crace.1/33 - (This is the shelf mark for a map held In the British Library's Grace Collection)	Manually
Conforms To	***6.4 conformsTo***	The names of standards that the digital object (digitized or born digital) complies with and which are useful for the use of the object.	0..*		string	**Optional data element**	W3C WCAG 2.0 (Language English)- (for an HTML document that conforms To web content accessibility guidelines)	Manually

Has Format	***6.5 hasFormat***	Use this element to identify another resource that is substantially the same as the digital object being described by the metadata but exists in a different format. Note that the purpose of this element is to give the identifier of the other resource in a different format, not to state the format of the object being described.	0..*		string	**Optional data element**	http://upload.wikimedia.org/wikipedia/en/f/f3/Europeana_logo.png (A link to another image format of the tiff image file being described).	Manually
Is Format Of	***6.6 isFormatOf***	Use this element to identify a related resource that is substantially the same as the digital object but in a different format. Use when there are alternative formats and it is not clear which preceded the	0..*		string	**Optional data element**	Europeana_logo.tiff (where the resource being described is a png image file)	Manually

		other.						
Has Version	***6.7 hasVersion***	A related object that is a version, edition, or adaptation of the described object. Changes in version imply substantive changes in content rather than differences in format.	0..*		LangString	**Optional data element**	The Sorcerer's Apprentice (Language English) - (for the translation by Edwin Zeydel of Goethe's poem Der Zauberlehrling, where the metadata record is describing the original).	Manually
Is Version Of	***6.8 isVersionOf***	A related object of which the described object is a version, edition, or adaptation. Changes in version imply substantive changes in content rather than differences in	0..*		LangString	**Optional data element**	opposite to 6.7	Manually

		format.						
Has Part	***6.9 hasPart***	A related object that is included either physically or logically in the described object.	0..*		string	**Optional data element**	Maps.added.22 231 - (The identifier for another map which is part of this one that is being described).	Manually
Is Referenced By	***6.10 isReferencedBy***	A related object that references, cites, or otherwise points to the described object.	0..*		LangString	**Optional data element**	Till, Nicholas (1994) Mozart and the Enlightenment: Truth, Virtue and Beauty in Mozart's Operas, W. W. Norton & Company (Language English)	Manually
References	***6.11 references***	A related object that is referenced, cited, or otherwise pointed to by the described	0..*		LangString	**Optional data element**	Security Magazine pp 3-12 (Language	Manually

		object.					English)	
Is Replaced By	***6.12 isReplacedBy***	A related object that supplants, displaces, or supersedes the described object.	0..*		string	**Optional data element**	http://dublincore.org/about/2009/01/05/bylaws/ (where the resource described is an older version, Say http://dublincore.org/about/2006/01/01/bylaws/)	Manually
Replaces	***6.13 replaces***	A related object that is supplanted, displaced, or superseded by the described object.	0..*		string	**Optional data element**	opposite to 6.13	Manually
Is Required By	***6.14 isRequiredBy***	A related object that requires the described object to support its function, delivery or coherence.	0..*		string	**Optional data element**	http://www.myslides.com/myslides.ppt (where the image being described is required for an online show)	Manually

Requires	***6.15 requires***	A related object that is required by the described object to support its function, delivery or coherence.	0..*		string	**Optional data element**	opposite to 6.14	Manually
	7. Collection	This category provides metadata information for logical groupings of contributed CH objects within a museum.						
Title	***7.1 title***	The title of the collection.	1..*		LangString	**Mandatory data element**	TNHM mineral collection images (Language English) Mineraalid Eesti Loodusmuuseumi kogudest (Language Eesti)	Manually
Creator	***7.2 creator***	This is the name of the creator of the collection.	1..*	use a consistent form of the name e.g.	Langstring	**Mandatory data element**	Rutt Hints	Manually

				Shakespeare, William.				
Subject	***7.3 subject***	This is the subject of the collection.	1..*		LangString	**Mandatory data element**	Earth science (Language English) Geology (Language English) Mineralogy (Language English) Geoteadused (Language Eesti) Geoloogia (Language Eesti) Mineraloogia (Language Eesti)	Manually
Description	***7.4 description***	A description of the collection.	0..*		LangString	**Optional data element**	Mineral specimen images from geological Collections of Estonian Museum of Natural History	Manually

							(Language English) Mineraalide pilte Eesti Loodusmuuseumi geoloogilistest kogudest (Language Eesti)	
Contributor	***7.5 contributor***	The name of contributors to the collection. This could be a person, an organisation or a service.	0..*	use a consistent form of the name e.g. Shakespeare, William.	LangString	**Optional data element**		Manually
	7.3.1 role	Kind of contribution.	1	**Please consult the MARC Code List for Relators, (available at id.loc.gov/vocabulary/relators.html) for possible roles.**	VocabularyTerm	**Mandatory data element IF 2.3 Contribute is used. Minimally, the Contributor(s) of the collection should be described.**		Default “Annotator” then manually

Type	***7.6 type***	The type of the collection as recorded by the content holder. Type includes terms describing general categories, functions, genres, or aggregation levels for content.	0..*	Preferably the values should be taken from the controlled DC Type vocabulary (provided Vocabulary)	LangString or VocabularyTerm	**Optional data element**	Image	Manually
identifier	***7.7 identifier***	This is the identifier for the collection	0..*		string	**Optional data element**	http://nla.gov.au/nla.pic-an7678346-1-v-cd	
Coverage	***7.8 coverage***	Coverage is the unqualified spatial or temporal coverage of the collection.	0..*		LangString	**Optional data element**	Boston, MA (Language English)	
	8. Europeana	This category provides various Europeana related metadata information needed for describing this CHO object.						

	8.1 dataProvider	The name or identifier of the organisation that contributes this object to Europeana. This element is specifically included to allow the name of the organisation who supplies this object to Europeana indirectly via an aggregator to be recorded and displayed in the portal. The name provided should be the preferred form of the name in the language the provider chooses as the default language for display in the portal.	1		string	**Mandatory data element.**	Estonian Museum of Natural History/ Eesti Loodusmuuseum	**Automatic**
	8.2 country	The name of the country of the data provider or "Europe" in the case of	1		string	**Mandatory data element.**	Estonia	**Automatic**

		Europe-wide projects						
	8.3 provider	The name of the organization that delivers data to Europeana. This is not necessarily the institution that holds or owns the original or digitised object. Where data is being supplied by an aggregator or project this element is the name of aggregator/project.	1		string	**Mandatory data element.**	The Natural Europe Project	**Automatic**
Licence	***8.4 licenceUri***	Information about copyright of the digital object as specified by 4.1 and 4.2. The value in this element is a URL that is constructed by adding a code indicating the copyright status of an object to the domain name	1	The values should be taken from the controlled vocabulary (provided)	URI	**Mandatory data element.**	http://creativecommons.org/publicdomain/mark/1.0/	Manually

		(europeana.eu domain or the creativecommons.org domain) where the status is defined.						

Example of Use for the ESE-CHO AP

The following indicates an example of the information kept for a digitised object following the ESE-CHO AP.

```
<collection xmlns="http://www.natural-europe.eu/nhm/aip/"
xmldc:dc="http://purl.org/dc/elements/1.1/" xmldc:dcterms="http://purl.org/dc/terms/">
  <header>
    <id>b50a161b-560f-44cd-a4fa-a8c6e2da80d3</id>
    <createdOn>2011-06-22T17:53:01.215+03:00</createdOn>
    <createdBy>tnhm2</createdBy>
    <lastModifiedOn>2011-06-22T18:11:28.682+03:00</lastModifiedOn>
    <lastModifiedBy>tnhm2</lastModifiedBy>
    <status>IN_USE</status>
  </header>
  <metadata>
    <dc:title xml:lang="eng">TNHM paleontology collection images</dc:title>
    <dc:title xml:lang="est">Fossiile Eesti Loodusmuuseumi kogudest</dc:title>
    <dc:creator>Rutt Hints</dc:creator>
    <dc:subject xml:lang="eng">Earth science</dc:subject>
    <dc:subject xml:lang="eng">geology</dc:subject>
    <dc:subject xml:lang="eng">paleontology</dc:subject>
    <dc:subject xml:lang="eng">evolution</dc:subject>
    <dc:subject xml:lang="eng">preihistoric life</dc:subject>
    <dc:subject xml:lang="est">geoteadused</dc:subject>
    <dc:subject xml:lang="est">geoloogia</dc:subject>
    <dc:subject xml:lang="est">paleontoloogia</dc:subject>
    <dc:subject xml:lang="est">evolutsioon</dc:subject>
    <dc:subject xml:lang="est">eelajalooline elu</dc:subject>
    <dc:description xml:lang="eng">
        Fossil specimen images from geological collections of Estonian Museum of Natural
        History
    </dc:description>
    <dc:description xml:lang="est">
        Kivististe pilte Eesti Loodusmuuseumi geoloogilistest kogudest
    </dc:description>
    <contributor role="Creator" roleuri="http://id.loc.gov/vocabulary/relators/cre">
        Rutt Hints
    </contributor>
    <type typeuri="http://purl.org/dc/dcmitype/Image">Image</type>
    <dc:identifier>http://nla.gov.au/nla.pic-an7678346-1-v-cd</dc:identifier>
  </metadata>
```

```xml
    <records>
      <record>
        <header>
          <id>2e12848f-8d8b-4b22-bffd-455f496d78bb</id>
          <createdOn>2011-06-22T20:20:01.046+03:00</createdOn>
          <createdBy>tnhm2</createdBy>
          <lastModifiedOn>2011-06-23T16:41:23.254+03:00</lastModifiedOn>
          <lastModifiedBy>tnhm2</lastModifiedBy>
          <version>1.2.alpha</version>
          <status>EDITING_IN_PROGRESS</status>
          <access>PRIVATE</access>
          <logs>
                  <log date="2011-06-22T20:20:01.046+03:00" actor="Tiit Hunt">
                      CONTENT_PUBLISHING
                  </log>
          </logs>
        </header>
        <metadata>
           <uri isLocal="false">http://geokogud.info/elm/specimen_image/g319/g319-
2_a.jpg</uri>
           <contextUri>http://geokogud.info/elm/specimen_image.php?id=690</contextUri>
           <resourceType>IMAGE</resourceType>
           <licenceUri>http://creativecommons.org/licenses/by-nc/3.0/</licenceUri>
           <sourceThumbnailUri>
http://147.27.41.103:8080/exist/rest//db/NHMRepository/content/thumbs/src/1klc1jqvrc9
bmhf7glun3oa7b9
          </sourceThumbnailUri>
          <dc:title xml:lang="lat">Mammuthus primigenius</dc:title>
          <classification uri="http://www.catalogueoflife.org/browse/tree/id/2362377">
              Mammuthus primigenius
          </classification>
          <contributor role="Photographer"
roleuri="http://id.loc.gov/vocabulary/relators/pht">
              Tiit Hunt
          </contributor>
          <subject uri="http://www.catalogueoflife.org/browse/tree/id/2362377">
             animalia
          </subject>
          <subject uri="http://www.catalogueoflife.org/browse/tree/id/2362754">
             chordate
          </subject>
          <subject uri="http://www.catalogueoflife.org/browse/tree/id/2362755">
               mammalia
          </subject>
          <dc:description xml:lang="eng">
```

```
            The molar tooth of woolly mammoth, dated back to about 10 000-10 500
            radiocarbon years is  one of youngest mammoth finds in Europe.
        </dc:description>
        <dc:description xml:lang="est">
            Puurmani lähistelt leitud karvase mammuti purihammas on Euroopa üheks
            noorimaks mammutileiuks. Selle vanuseks on määratud 10 000-10 500
            radiosüsiniku aastat.
        </dc:description>
         <dc:publisher xml:lang="eng">Estonian Museum of Natural History</dc:publisher>
         <dc:publisher xml:lang="est">Eesti Loodusmuuseum</dc:publisher>
         <type xml:lang="eng">subfossil</type>
         <type xml:lang="est">subfossiil</type>
         <dc:format xml:lang="eng">image/jpeg</dc:format>
         <dc:identifier>
             http://geokogud.info/elm/specimen_image/g319/g319-2_a.jpg
         </dc:identifier>
         <dc:rights xml:lang="eng">
               Copyright © Estonian Museum of Natural History
         </dc:rights>
        <dc:rights xml:lang="est">Autoriõigus © Eesti Loodusmuuseum</dc:rights>
        <dc:source xml:lang="eng">The Natural Science Magazine pp 107-234</dc:source>
        <dcterms:issued>1920</dcterms:issued>
        <dcterms:alternative xml:lang="eng">woolly mammoth</dcterms:alternative>
        <dcterms:alternative xml:lang="est">karvane mammut</dcterms:alternative>
        <dcterms:extent xml:lang="eng">3000 × 2400 pixels</dcterms:extent>
        <dcterms:isPartOf xml:lang="eng">
            Collection of mammoth molars
        </dcterms:isPartOf>
        <dcterms:isPartOf xml:lang="est">Kogu mammuti purihambaid</dcterms:isPartOf>
        <dcterms:spatial xml:lang="eng">Puurmani</dcterms:spatial>
        <dcterms:spatial xml:lang="eng">Estonia</dcterms:spatial>
        <dcterms:spatial xml:lang="est">Puurmani</dcterms:spatial>
        <dcterms:spatial xml:lang="est">Eesti</dcterms:spatial>
        <dcterms:temporal xml:lang="eng">Pleistocene – Holocene</dcterms:temporal>
        <dcterms:temporal xml:lang="eng">Pleistotseen - Holotseen</dcterms:temporal>
      </metadata>
    </record>
  </records>
</collection>
```